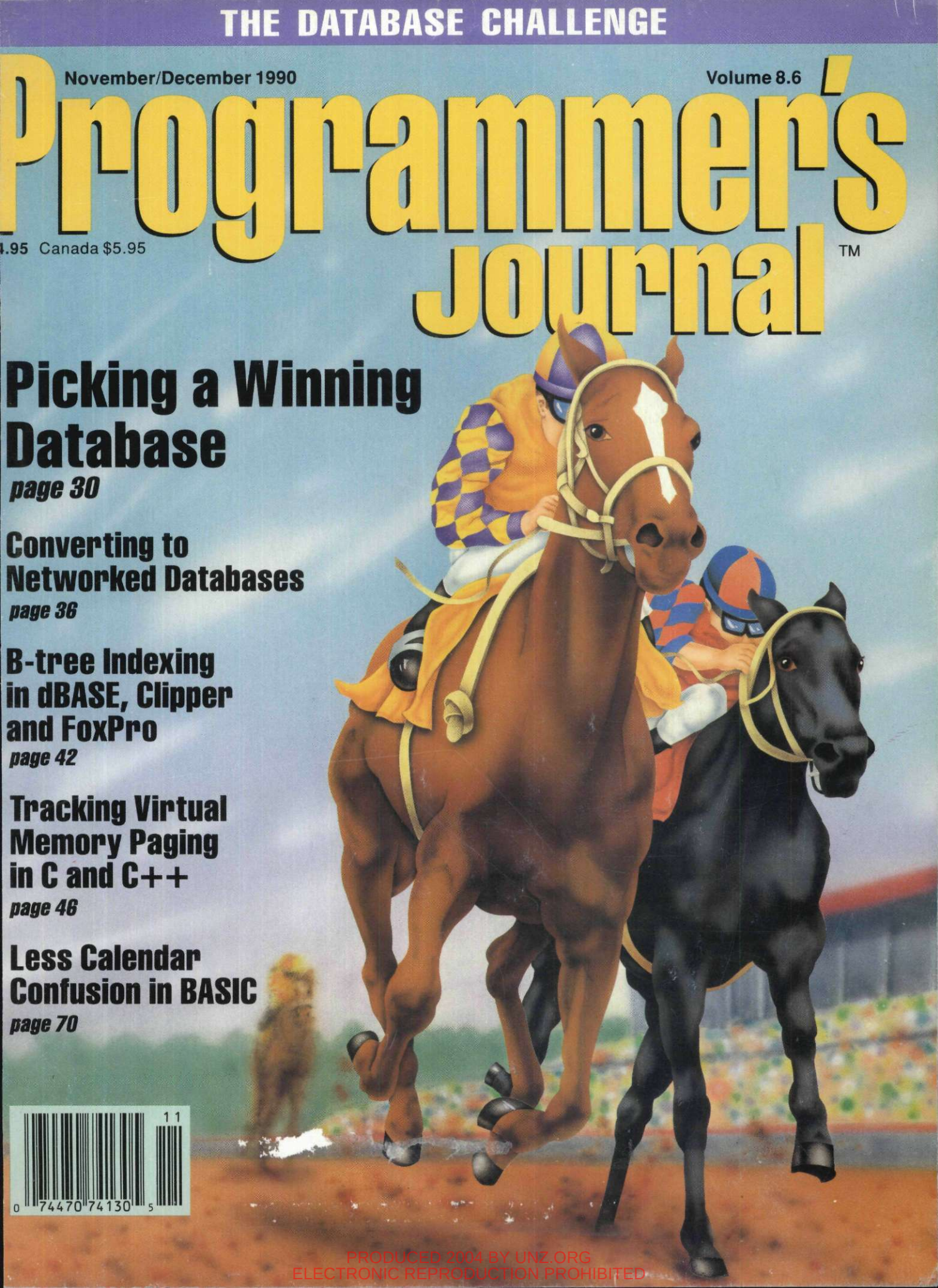
THE DATABASE CHALLENGE

November/December 1990 Volume 8.6

Programmer's Journal™

4.95 Canada $5.95



Picking a Winning Database
page 30

Converting to Networked Databases
page 36

B-tree Indexing in dBASE, Clipper and FoxPro
page 42

Tracking Virtual Memory Paging in C and C++
page 46

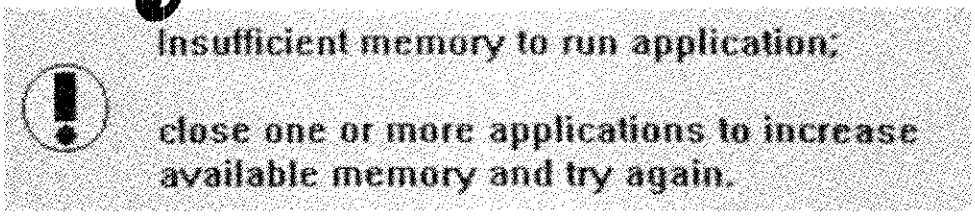
Less Calendar Confusion in BASIC
page 70

11
0 74470 74130 5

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

386 and 486 Windows users:

DOS Lives. Despite what you may have read.



Windows 3.0 may have been a big step forward for some programs.

But it was a big step backward for DOS. Suddenly, it was 1987 all over again. Not enough room for DOS programs to run because TSRs, utilities, drivers and buffers were taking up room your DOS programs need.

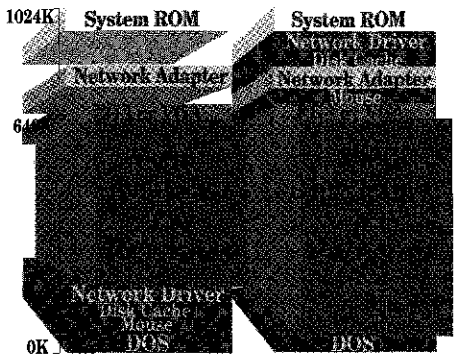
When Windows 3.0 says there's not enough room to run your DOS programs, it's just trying to tell you it needs QEMM 5.1.

QEMM 5.1 to the rescue

Now, we've updated QEMM to provide additional memory for DOS programs within Windows 3.0.

QEMM 5.1 works with the built-in capabilities of your 80386 or i486 processor to find and recover unused memory segments. As you can see in the chart, there are gaps in your PC's memory usage above 640K. QEMM 5.1 fills those gaps and provides more room for your DOS programs to run.

And of course, QEMM 5.1 still works with DOS when you're not running Windows. You get all the same benefits: up to 130K more memory to run the new generation of memory-hungry programs; space for larger spreadsheets and database files.



Before *After*

Unused memory is like hidden gold in your PC. QEMM finds it and makes it available.

Introducing DESQview 2.3 and DESQview 386 2.3 for users of Windows 3.0

They said it couldn't be done, but DESQview 2.3 can run Windows 3.0 programs. Not just in Windows "Real mode" but in "Standard" mode. That means programs can be up to 16MB.

And it can run DOS programs and DOS-extended programs i.e., 1-2-3 Release 3, side-by-side.

DESQview 386 2.3 does all that and more. It lets you run 386 DOS extended programs like AutoCAD 386 and IBM Interleaf side-by-side.

DOS, extended DOS; Windows—whatever standards you set, we will support. We're committed to helping you get the most out of your hardware and software today. And tomorrow.

QEMM-386 System Requirements: 80386-based PCs and PS/2s and PCs with 80386 add-in boards. Operating system: PC DOS 2.0-4.0, MS DOS 2.0-3.3, Windows 3.0. Conventional memory requirement 1.5K.
DESQview System Requirements: IBM Personal Computer and 100% compatibles (with 8086, 8088, 80286, or 80386 processors) with monochrome or color display; IBM Personal System/2 • Memory: 640K recommended; for DESQview itself 0-145K • Expanded Memory (Optional): expanded memory boards compatible with the Intel AboveBoard; enhanced expanded memory boards compatible with the AST RAMpage; EMS 4.0 expanded memory boards • Disk: two diskette drives or one diskette drive and a hard disk • Graphics Card (Optional): Hercules, IBM Color/Graphics (CGA). IBM Enhanced Graphics (EGA), IBM PS/2 Advanced Graphics (VGA) • Mouse (Optional): Mouse Systems, Microsoft and compatibles • Modem for Auto-Dialer (Optional): Hayes or compatible • Operating System: PC-DOS 2.0-4.0; MS-DOS 2.0-3.3 • Software: Most PC-DOS and MS-DOS programs; programs specific to Microsoft Windows 1.03-2.1, GEM 1.1-3.0, IBM TopView 1.1 • Media: DESQview is available on either 5-1/4" or 3-1/2" floppy diskette.

Trademarks: Windows, MS-DOS: Microsoft Corporation; PS/2, Interleaf, TopView: IBM Corporation; 80386, i486, AboveBoard: Intel Corporation; 1-2-3: Lotus Development Corporation; AutoCAD 386: Autodesk, Inc; RAMpage: AST Research; Hercules; Mouse Systems; Hayes; GEM: Digital Research, Inc.

©1990 Quarterdeck Office Systems

Yes! I need increased DOS productivity in Windows!

Qty	Product	5-1/4	3-1/2	Each	Totals
	QEMM 386 5.1			$99.95	
	DESQview 386 2.3 (includes QEMM)			$219.95	
	DESQview 2.3			$129.95	
	Shipping & Handling $5 in USA/$10 outside USA				
	California Residents add 6.75%				
	Grand Total				

Payment method ☐ Visa ☐ MasterCard
Expiration ____/____
Card # _____
Name _____
Address _____
City _____
State ____ Zip _____

Please allow 3 weeks for delivery

Quarterdeck

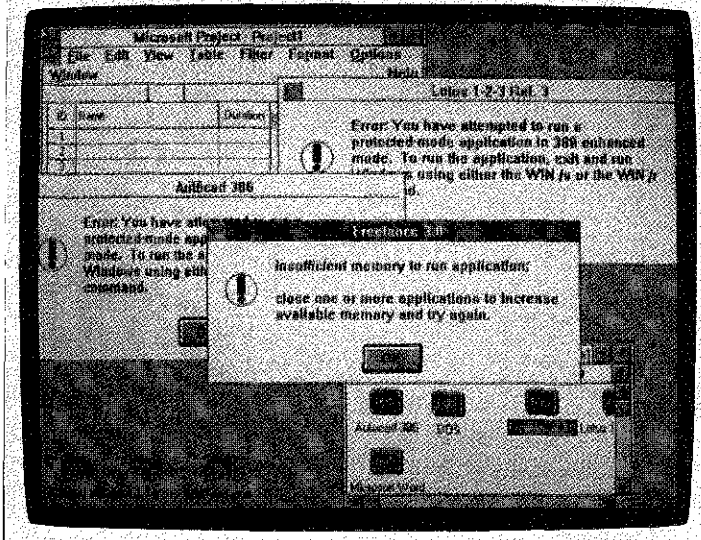
Quarterdeck Office Systems, 150 Pico Blvd., Santa Monica, CA 90405 (213) 392-9851 Fax: (213) 399-3802

Circle 147 on Reader Service Card

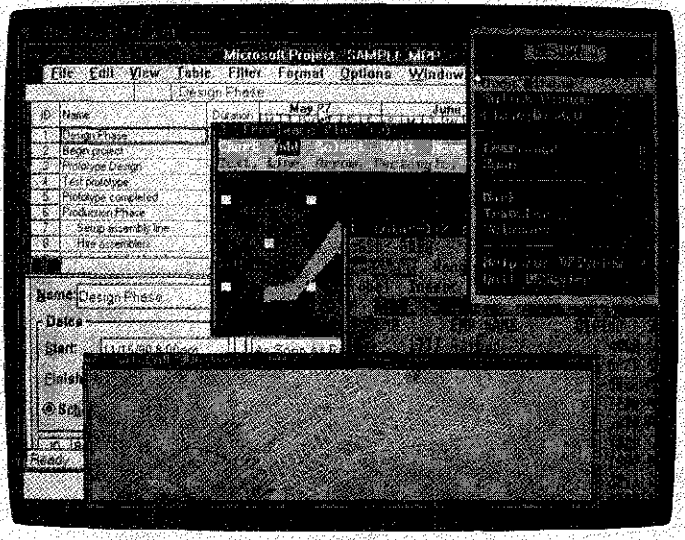
PRODUCED 2008 BY SOFTARG
ELECTRONIC REPRODUCTION PROHIBITED

Is Windows 3.0 the end of DESQview™?

Not if you're still using DOS programs.


Microsoft Project
File Edit View Table Filter Format Options
Window
Error: You have attempted to run a protected mode application in 386 enhanced mode. To run the application, exit and run
Insufficient memory to run application;
close one or more applications to increase available memory and try again.
OK

Windows 3.0. The multitasking, windowing environment.



DESQview 386 2.3. The multitasking, windowing environment.

Are you still using MS-DOS programs on your PC?

You may want to use DESQview as your primary operating environment.

The new DESQview 2.3 and DESQview 386 2.3 let you use your favorite DOS and DOS-extended programs in windows side-by-side on 80286, 80386 and i486 PCs. As you can see above, you can even run Windows programs within DESQview 386. So the next time you get error messages like the ones at the left in Windows, remember how the same set of programs look running in DESQview.

Whatever standard you use—DOS, extended DOS or Windows—DESQview is still the best way to get the most out of the hardware and software you own today.

DESQview. The obvious choice.

Quarterdeck

Quarterdeck Office Systems, 150 Pico Boulevard, Santa Monica, CA 90405 (213) 392-9851 Fax: (213) 399-3802

This comparison was made using a system like the one you might run: Both shots show an ALR FlexCache 33/386 running DOS 3.3 with VGA display adaptor, Novell NetWare v3.01 Rev. A, with IPX/SPX v3.01 Rev A, Microsoft Mouse 7.00, and Microsoft SMARTDrive v3.03 disk cache. Buffers were set to 20. For the Windows screen, we ran Microsoft Windows 3 HIMEM.SYS and EMM386.SYS. For the DESQview screen, we ran QEMM 386 v5.1.

Trademarks: Microsoft, Windows, MS-DOS, 80386, i486, ALR, FlexCache, Novell, NetWare. ©1990 Quarterdeck Office Systems

Circle 147 on Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Programmer's Journal™

November/December 1990 Volume 8.6

FEATURES

ON GRAPHICS **VGA COLOR PAGING** *By Michael Abrash* 20
Installment 21, in which Michael explores the complexities of VGA color paging

DATABASE **THE DATABASE DILEMMA** *By Brett Glass* 30
Brett provides insights into choosing your database programming environment

DATABASE **NETWORKING DATABASES** *By Lea C. Curtis* 36
Converting applications from single-user to multi-user in dBASE languages

DATABASE **B-TREE INDEXING** *By M. Steven Baker* 42
Exploring implementation tradeoffs in dBASE, Clipper, and FoxPro indexing

C/C++ **VIRTUAL MEMORY PAGING** *By Allen Holub* 47
Allen continues his exploration of virtual arrays, focusing on paging strategies

C **WRITING AN ANSI C PREPROCESSOR** *By W. Sumerlin* 58
An overview of implementation problems in complying with the ANSI C standard

386 NOW **PEAK PERFORMANCE: ON TO THE 80486** *By Mark Barrenechea* 64
First in a series of articles on compiler optimization, starting with a look at instruction scheduling and 80486 parameter passing options

BASIC **CALENDAR CONVERSIONS** *By Murray L. Lesser* 70
Murray corrects a number of notions about calendars (Julian, Gregorian, etc.)

ADA **SOFTWARE COMPONENTS** *By Richard Riehle* 75
Component-oriented software engineering and reusability in ADA

ASM **OPTIMIZATION STRATEGIES** *By John Otken* 79
John details optimization strategies for your next assembly language program

COLUMNS

REDIRECTIONS **PINK NOISE** *By Frank Greco* 14

OFFRAMP **OH SAY CAN YOU C?** *By Hal W. Hardenbergh* 96

DEPARTMENTS

EDITOR'S NOTE 4

LETTERS 6

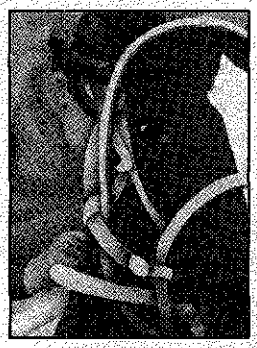
PJ PRESSROOM 88

PJ BBS LISTING 41

PJ BILLBOARD 93

ADVERTISER INDEX 95

COVER



It takes good horse sense to choose the right database environment. Whether you're out for a casual stroll or running neck-and-neck for the roses, this issue will help you reach the finish line in record time.

Artist: Rob Schuster

Articles and letters in *Programmer's Journal* represent the views of the authors and not necessarily those of the publisher. Letters to the editor become the property of *Programmer's Journal* and may be edited for publication.

Note that all code and listings in *Programmer's Journal* appear "as is," and *Programmer's Journal* makes no guarantee or warranty of their quality, performance, accuracy, or suitability for any particular purpose. *Programmer's Journal* assumes no responsibility or liability to any person or entity with respect to any damage caused or alleged to be caused by their use.

Advertising: For ad rates, deadlines, and placement, call Robert Tappan or Jay Moore at 800-234-0386, or write to *Programmer's Journal*, P.O. Box 70167, Eugene, OR 97401-0110. *Programmer's Journal* is produced using the following software products: Ventura Publisher,™ Aldus PC PageMaker,® and Corel Draw. The PostScript™ files are output on a Linotronic™ 200.

Programmer's Journal (ISSN 0747-5861) is published bimonthly by Oakley Publishing Company, 150 N. 4th Street, Springfield, OR 97477-5454. Second-class postage paid at Springfield, OR, and additional mailing offices. USPS # 000293. **POSTMASTER:** Send address changes to *Programmer's Journal*, PO Box 70387, Eugene, OR 97401-0121.

© 1990 OAKLEY PUBLISHING COMPANY

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

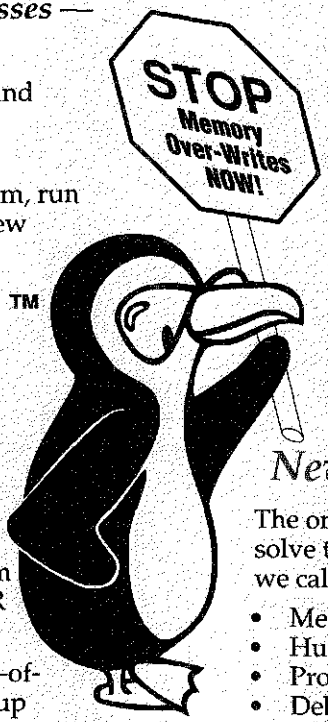
POWER DEBUGGING

BOUNDS-CHECKER

Finds out-of-bounds memory accesses — AUTOMATICALLY.

Flush out those Nasty pointer problems and other out-of-bounds memory accesses — AUTOMATICALLY.

Each time you make a change to a program, run BOUNDS-CHECKER while testing the new code. If you accidentally access out-of-bounds memory, BOUNDS-CHECKER will pop up displaying the offending SOURCE LINE. And your program runs at full speed.



"BOUNDS-CHECKER and Soft-ICE make sophisticated use of the most powerful versions of Intel's processor family to track down some of DOS programming's most insidious bugs. If you're developing programs for DOS, these are essential tools."
PC Magazine
July, 1990 pg. 48

Ship Bug-Free Products

You can run BOUNDS-CHECKER while testing your program. There are no additional steps to your testing cycle, but you can feel secure when the program has passed through BOUNDS-CHECKER with no reported problems.

Many over-write problems and other out-of-bounds memory accesses do NOT show up during normal testing. An out-of-bounds memory location may be modified, but that particular location doesn't happen to be important at the time. Once the program is in the field and a certain network is loaded or a certain T&SR or device driver is loaded, that memory location suddenly becomes very important... AND THE SYSTEM CRASHES.

You can prevent these problems by making BOUNDS-CHECKER a standard part of your testing procedure.

Gives you the protection of a protected operating system under MS-DOS.

BOUNDS-CHECKER uses the 80386 virtual machine technology to provide real-time memory protection. In addition BOUNDS-CHECKER uses the symbolic information output by your compiler to differentiate CODE and DATA. When your program is running, BOUNDS-CHECKER protects the program's CODE and all memory outside your program.

Requires 80386 PC.
MS-DOS is a trademark of Microsoft Corporation.

Soft-ICE 2.5
New Version, New Features

The only debugger specifically designed to solve those problems unique to MS-DOS that we call the DOS Nasties.

- Memory over-writes
- Hung programs
- Program too big to debug
- Debugging T&SRs and Loadable Drivers
- Multiple Symbol Tables
- Supports Microsoft C 6.0 & Turbo C++

MagicCV 3.0
New for C6.0

A set of tools designed to ease the memory crunch with Microsoft C 6.0.

- Run Code View in ***Less Than 8k***
- Run CodeView with EMM & VCPI
- Increase heap space when compiling
- Increase memory with make
- Load high T&SR's and device drivers
- VCPI support

BOUNDS-CHECKER ***$249***
Soft-ICE 2.5 ..... ***$386***
MagicCV 3.0 ..... ***$199***

Special Offer...

Buy BC & S-ICE ..... ***Save $100***
Buy S-ICE & MCV ..... ***Save $86***
Buy all three ..... ***Save $186***

30 Day Money-Back Guarantee

Nu-Mega TECHNOLOGIES

CALL TODAY (603) 888-2386 or FAX (603) 888-2465
P.O. BOX 7607 ■ NASHUA, NH ■ 03060-7607 ■ U.S.A.

Circle **160** on Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

EDITOR'S NOTE

As I write this in September, the heat of summer has given way to shorter autumn days. My plunge into the river waters is a bit more invigorating, and will soon be downright bracing. In the computer world, the quieter summer pace is gearing up to the frenzy of fall COMDEX and new product introductions. If you've never attended a COMDEX show, you've missed experiencing the power and wealth of the PC marketplace in its full force.

THE DATABASE CHALLENGE

This issue focuses on database design and programming. Databases probably account for more hours of programming and consulting than any other application. By their nature, database applications almost always require customization. While academic theory indicates the supremacy of the "relational" model (SQL) over other models ("hierarchical" used in mainframe SIR and "network" implemented in Raima's db Vista PC products), most PC database programming still uses dBASE-compatible file formats.

The original dBASE is closer to a flat file manager than any of the three common academic database models are. The relational features of dBASE are there, but they have to be programmed in by hand. This, in turn, supports a large add-on market for programmer tools and libraries. The success of SQL engines in the workstation market is not yet being repeated on the PC. Although SQL use may be gaining acceptance, the number of applications using dBASE-compatible file formats and languages is also increasing as the PC world continues on its ever-widening track.

Regardless of academic family, a major consideration in PC database programming is network support. Some products (IBM's OS/2 Extended Edition Database Manager, Novell's Btrieve, and Borland's Paradox, for example) were intentionally designed for networks. Others using the dBASE language (dBASE, Clipper, FoxBASE, etc.) appear to have network capability as an afterthought. Converting a single-user application to a networked version can require considerable work, hence the advantage of using SQL-based query languages that can handle network considerations more transparently. In response to the demand for software that operates over a network, a number of vendors are adding SQL support to their other programming products (such as Microsoft C and BASIC compilers and Borland's Paradox Engine), and we should expect to see more of this. The continuing problem with DOS networks, however, is that networking software is always competing with the database manager for that precious 640K space.

NO WINTER DOLDRUMS

Aside from a rather resplendent Windows 3.0 announcement and some potentially problematic lawsuits, the PC industry has had a relatively quiet summer while I've been enjoying my swims in the river. However, the once-clear PC waters are being muddied by some long-term transitions. The "range wars" continue for the PC graphics user interface (GUI) and for the operating environment of choice. Will it be Windows 3.0, OS/2 of some flavor, DesqView/X, or one of the Unix GUIs? The Unix community is experiencing a similar horse race between SUN/AT&T's "Open Look" and "OSF/Motif."

OS/2, once considered the natural heir to DOS, is experiencing its own range war between Microsoft and IBM. The market for OS/2 has been severely damaged by internal disagreements which have finally surfaced in public. OS/2 2.0 (386 version) will undoubtedly be delayed until late next year. As Intel goes into full-scale production of the 80486, the PC community is still debating—embarassingly—about software several cycles behind. Out of all this confusion, someone will eventually emerge as the winner. Unfortunately, it isn't likely to be the programmer/developer, who has to rely on guessing right.

Steven Baker

Programmer's Journal

Editor
M. Steven Baker
Technical Editor
Colin Elligsen
Associate Editors
Cecelia Hagen
Deborah Healey
Contributing Editors
Michael Abrash
Frank D. Greco
Tom Swan

Publisher
Elizabeth Oakley
Associate Publisher
Bobbi Sinyard
Accounting
Jill Edwards
Circulation
Barbara Curtright
Subscriptions
Pamela Coleman
Office Manager
Sylvia Long
Administrative Assistant
Randy Randolph
Production & Art Director
Wanda Walker
Production Artist
Nicole Harris
Illustrations
Jim Carpenter

Sales & Marketing Director
Robert M. Tappan
Advertising Manager
Jay Moore

Business and Editorial Offices
Programmer's Journal
P.O. Box 70167
Eugene, OR 97401-0110
(503) 747-0800
(800) 234-0386
Fax: (503) 746-0071
BBS: See page 41

Subscriptions
1 year (6 issues) $24.00

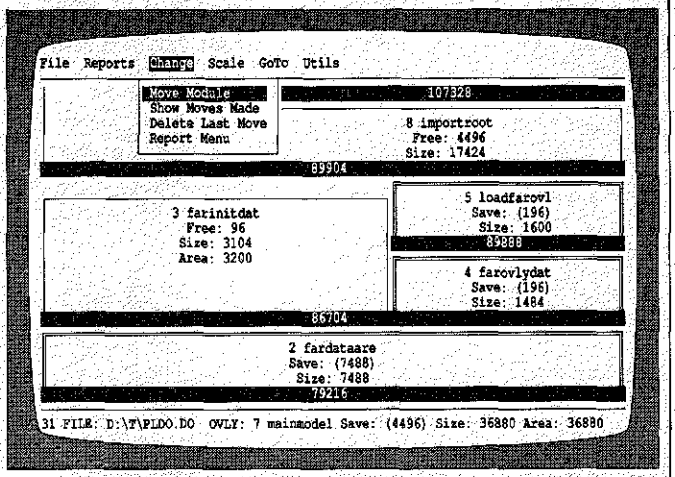
Change of Address
Programmer's Journal
PO Box 70387
Eugene, OR 97401-0121

© 1990 OAKLEY PUBLISHING COMPANY

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

OVERLAY DESIGNER™

for Plink86 and RTLink



What You See Is What You Linked

Save Programmers' Time

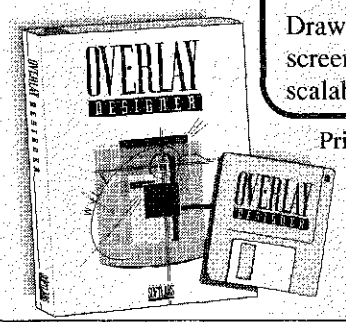
You can expect to save up to 75% of the time you now spend analyzing and testing overlay structures, and still do a better job. Overlay Designer is a screen-oriented data base and presentation program to help programmers design and analyze the overlay structures of large DOS application programs. Overlay Designer ***automatically reads the .MAP file, .OBJ files, .LIB files***, and (for RTLink only) the monitor output captured to file during the link, and builds a database describing the link. The programmer then uses Overlay Designer's graphic, algorithmic, and tabulation tools to design a compact, fast, and reliable overlay structure. Overlay Designer is ***Computer Aided Design*** for DOS overlays. Overlay Designer has been beta tested for 9 months on products with sales from one million dollars per year to one million dollars per month.

Fast Results - Almost No Setup

You can select 3 types of data to import into the Overlay Designer file: 2-dimension overlay diagram, detailed memory sizes, and symbol references. The diagram import takes only a few seconds. One beta test programming team posted a 3 ft x 4 ft overlay diagram on the wall each Friday. The full import takes less time than the link.

A Fully Professional Software Package

Overlay Designer has over 120,000 lines of source code, developed by the primary author of Q&A (® Symantec Corp., InfoWorld Product of the Year 1987, 1988, PC Magazine Technical Excellence Award 1988), and author of Spellguard (® Computer Associates, Infoworld Product of the Year 1981). It is fully documented with an indexed 110 page manual and has context sensitive help at every input screen. Many early adopters ***use Overlay Designer regularly*** and find it an ***indispensible tool*** for overlay design. Overlay Designer is recommended for links with more than 25 overlays, where products that claim to design overlays automatically prove ineffective and simulated virtual memory is too inefficient.



Benefits

- Programmer time savings give ***quick return on investment***
- Diagrams expedite communications in multi-person projects
- Improve morale by ***eliminating tedious detail work***
- Meet project deadlines by reducing overlay design time
- ***Reduce QC testing***; many overlay bugs found automatically
- Improve your product with compact, reliable overlay
- ***Automatic technical documention*** of overlays

"After my first day with Overlay Designer, I had done work that would have otherwise taken me 2 weeks."
- Mike Milsner, Senior Programmer, Symantec Corp.

Features

- Detection of direct ***"equal level" overlay*** calls
- 35 dynamic tables to cross reference symbols & their use
- Separation of references into the categories ***code*** and ***data***
- Scalable screen and hard copy diagram of overlay structure
- Laser printer overlay diagrams ***scaled to memory sizes***
- Free area sizes and potential memory reductions on diagram
- ***"What if" testing of moving memory*** between overlays
- Sort operations on tables to identify important relationships
- Filter operations on tables to eliminate extraneous data
- ***Run editor program*** without leaving Overlay Designer
- Memory size reports by segdef, module, file, & overlay
- Memory sizes reported in hexidecimal or decimal
- ***Stack report tables*** up to 12 deep
- Immediate hard copy of all or part of any table
- Cross-reference publics & externals by file, module, overlay

Introductory Offer: ***Overlay Diagrammer.***™
Draws and labels overlay diagrams instantly - screen and hard copy. Laser printer diagrams are scalable from 3 inches x 3 inches to 7 feet x 7 feet.

Pricing: Includes 60 day warranty & 1 year support.
Overlay Designer: 1 user license $595
4 user license $1,190
Overlay Diagrammer: 1 user license $149
(90% credited toward Overlay Designer purchase in 3 months.) 4 user license $298

Plink86 ® Sage Software. **RTLink**™ Pocket Soft, Inc.

COLEMAN SOFTLABS INC

Coleman Softlabs, Inc.
296 Bay Road,
Atherton, CA 94027
Phone (415) 322-9006
Fax (415) 322-1622

Circle **39** on Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

LETTERS

FROM RUSSIA WITH ... HEAPS

Turbo Pascal 4.0+ heap managers have been thoroughly discussed in *Programmer's Journal*. Articles such as "Heap of trouble?" (*PJ* 6.3), "How many holes does it take to fill an IBM PC?" (*PJ* 5.1) and "Programming on the variant express" (6.5) by Tom Swan are all worth mentioning.

I have some thoughts on this topic. Tom Swan suggested a perfect method for using as much heap space as your data needs without exceeding the size of the dynamically allocated string variable. But what method do you use to extend such a variable? This is a rather general situation while editing or entering data. Let us define some types for handling such resizeable variables:

```
type     VarStrPtr   = ^VarStr;
         VarStr      = record
                         MaxLen : byte;
                         TheStr : string;
                       end;
```

We also need to define our allocation and deallocation procedures:

```
procedure NewStr(var P: VarStrPtr;
 StrLen: byte);

var i: integer;
begin
    i := (((StrLen + 1) div Quantum)+1) * Quantum;
    if i > 256 then
       i := 256;
    GetMem(P, i);
    P^.MaxLen := i-1;
 end;
procedure DisposeStr(var P: VarStrPtr);
 begin
    FreeMem(P, P^.MaxLen+1);
    P := nil;
 end;
```

If your string was allocated by the **NewStr** procedure, you can increase its length, but not by more than *MaxLen* specifies. If you wish to expand it considerably, you need to allocate a new *VarStr* record that uses more memory, copy your string to the new variable, and dispose of the old record.

The effectiveness of this method depends on the value of the constant *Quantum*. If it is small, you will allocate, copy, and deallocate strings very frequently. This can waste time and fragment heap memory. If it is very big, you will waste a lot of heap memory.

It is not advisable to use *Quantum* < 8 because of internal details of TP4+ free-space list storage. When you deallocate variables shorter than 8 bytes, you can actually decrease the available heap space.

ILLUSTRATION BY JIM CARPENTER

If $n>8$ you can free some heap space when your string length becomes lower than *MaxLen* - *n*. To free space, allocate a shorter string, copy your string to the new record, and deallocate the old record. You can free *n* bytes with each operation. Since this operation is optional, you may prefer to waste some bytes in order to save a few milliseconds.

If you use this technique, your application with *m=VarStr* records wastes approximately $(m*n)/2$ bytes.

Oleg P. Pilipenko
Odessa, USSR

KUDOS TO ABRASH

I haven't finished *PJ* 8.5 yet, but I was so impressed by Michael Abrash's column on VGA arcana that I had to write. I am no great graphics aficionado, but I *do* have one of these things on my machine, and I *had* noticed it would flicker and flash during some otherwise-attractive demo programs.

I had pretty much traced it down to waiting for the vertical re-trace before fiddling with the registers, but Abrash's column is the *first and only* mention of this I have *ever* seen in print—although judging by his comments and my experience, it's not that uncommon. Nary a peep from those five million VGA card reviews in the power-user magazines. Amazin', ain't it?

Aside from defensive programming practices, *purchasers* of VGA cards may want to try a BBS program like DAZZLE, which is a beautiful and lovely thing, and snows on my unit like mad. An inexpensive commercial program which similarly misbehaves is FRACTAL MAGIC. These programs were obviously written on equipment which did *not* snow, so it's quite apparent that there are wide variations which, aside from Abrash's helpful column, are otherwise not-to-be-spoken-of.

One thing Abrash didn't mention—I suppose the implication is pretty obvious—is that waiting for the vertical interval *takes time*, meaning that programs that really want to bang the screen at a high rate of speed will be inherently incompatible with some VGA adaptors.

J.G. Owen
Fort Salonga, NY

OFFRAMP—NO EXIT?

Your "Offramp" column is off to an inauspicious start in the July/August issue (*PJ* 8.4). Not only does Mr. Hardenbergh exhibit a typically paranoid attitude toward Japanese business, but by citing everything from glasnost and social policy to quality circles and the financial markets, he tries unsuccessfully to convince the reader that he is better read than his audience and that his views should therefore be assumed to be correct simply because of the extensive research his myriad citations imply.

I am not going to try to present a complete rebuttal of Mr. Hardenbergh's sparse argument, but I would like to correct a few problems in his analysis:

First, capital costs are equivalent to the cost of business loans *only* in firms that have no other source of capital. The cost of capital for companies in the chip fab business is significantly different from, and usually lower than, the rates on business loans. The situation is more complex than Mr. Hardenbergh thinks. To claim that our cost of capital would match that of the Japanese if only Congress made savings interest non-taxable is naively simplistic.

Second, Congress cannot fix all our industrial and economic woes overnight just by repealing the Sherman Act or changing the tax code. As in physics, every legislative action provokes many reactions. If Mr. Hardenbergh asked an economist what would happen if any of his "solutions" came to pass, I think he would stop being quite so righteous.

Third, a major reason that U.S. companies lost the lead in DRAM manufacture was not because of a diabolical ploy by MITI, but because they *chose* to. Early in the game Japanese companies concentrated on the *manufacture* of chips, while their competitors in the U.S. concentrated on *design*. The Japanese perceived (correctly) that chips would

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

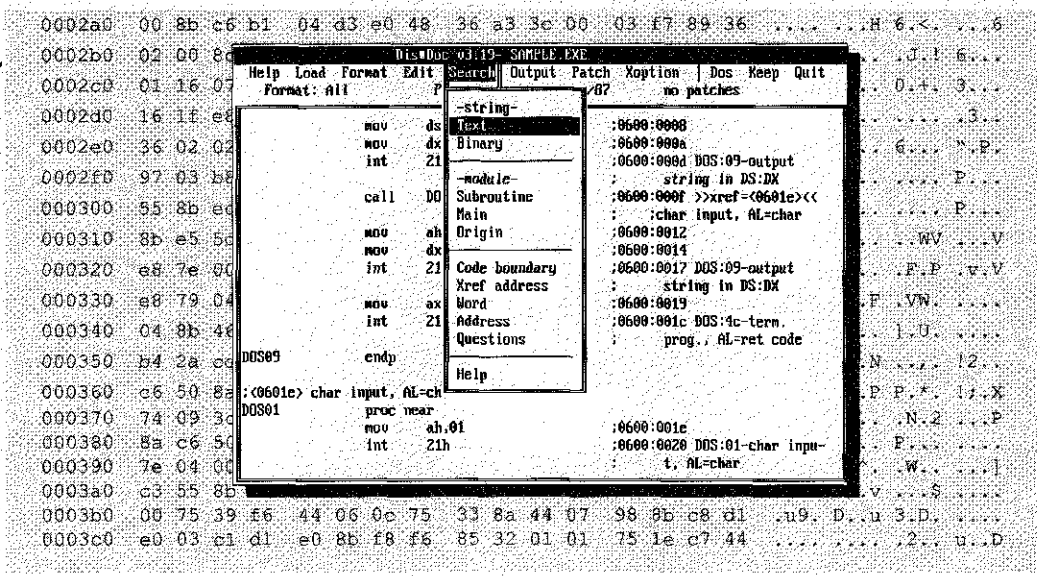
NEW VERSION

No Source?
No Problem.

Automatic BIOS, PSP, MS-DOS interrupt labeling built-in

Batch and Interactive modes

Interactively locates code/data boundaries



8086 to 80486 instruction sets supported

Interactive cross-referencing

Creates listing only after disassembly is perfect

Dis•Doc Professional™

New **Dis•Doc** *Professional* is your **dual-mode** key to any DOS source code. It works in batch and interactive modes simultaneously, allowing you to generate the core information of even the most complex programs **fast**...and modify them even faster. Most programs will come apart in just two minutes. Imagine what you can do with a tool this powerful! **Dis•Doc** sifts through programs **eight times** for guaranteed accuracy. When code gets mixed up with data, our toolbox comes to the rescue with **smart** search and **easy** edit utilities.

Warning: Dis•Doc *Professional* may change the way you work **forever**.

Programmers who used to shy away from fixing outmoded programs with no source code are going to discover a **valuable** new talent: the ability to modify and revise codes that would cost way too much to start over (it's a programming manager's dream). Save your employer huge new-programming fees and enhance your marketability. **Dis•Doc** is so easy to learn, you'll be a high-dollar hero in no time!

Dis•Doc *Professional* is an amazing new teaching tool. Learn how programs work...take them apart and see the writing techniques that top pros use. Use it to assist in debugging. Hunt down viruses and write killers. **Dis•Doc** can save you **years** of frustration, and it only costs $249.95.

COMPARE FOR YOURSELF!

	Dis•Doc™	Sourcer™
Speed of disassembly (bytes/sec)	***822***	*120*
Accuracy (code/data separation*)	***99.9%***	*99.9%*
Unlimited file size	***YES***	*NO*
BIOS labeling	***YES***	*XTRA $$*
BIOS/PSP file labeling	***YES***	*NO*
Unpacks EXE files	***YES***	*XTRA $$*
Patcher	***YES***	*NO*
Batch mode	***YES***	*YES*
Interactive mode	***YES***	*NO*
Time it takes to customize a label	***5 sec.***	*2-5 min.*
Time it takes to change data to code	***5 sec.***	*2-5 min.*

*at eight passes based on a 9040 byte EXE file picked at random

To order your **Dis•Doc Professional**™ disassembler toolkit or our $3 demo disk, simply call:

1-800-336-1961 in US
1-203-953-0236 outside US
or send check or money order for **$249.95** plus $5 s&h to:

RJSwantek, Inc.
178 Brookside Road
Newington, CT 06111-1320 USA

For more information call 1-203-953-0236 • FAX available • MasterCard & Visa • Shipped Immediately Via UPS Blue inside US.

Sourcer is a registered trademark of V Communications, Inc.

Circle **59** on Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

become commodity items and aggressively pursued that course. They applied the learning curve concept with a vengeance (a concept that was first described by the Boston Consulting Group and used successfully in 1963 by Fairchild Semiconductor to sound the death knell for the vacuum tube market). Their U.S. competitors, meanwhile, banked on chips remaining high value-added items, and continued to command hefty margins for their products.

These companies made important strategic decisions. As it turned out, the Japanese companies were right about DRAMs, while the U.S. companies were right about more specialized chips like the Intel 80x86 and the Motorola 68000 series, which is about how the semiconductor market has played out.

In the case of Intel, sure they invented the DRAM, but they also invented plenty of other silicon goodies. When faced with the growing commoditization of the DRAM market, they wisely reallocated their resources to the microprocessor and other markets where their prospects of making money were far greater. Simply having invented DRAMs is no reason to continue making them when better alternatives exist.

I suggest Mr. Hardenbergh read George Gilder's *Microcosm* if he wants to find out what is *wrong* with the semiconductor industry. If he does, I think he will agree that the situation is much more complex than he realizes and that what is wrong has little resemblance to his xenophobic fears. In fact, I think he will agree that there is nothing wrong at all in either the conspiratorial or the national-security sense. The Japanese are simply players in these dynamic (no pun intended) world-wide markets, and not the sly, imperialistic threat that Mr. Hardenbergh seems to see.

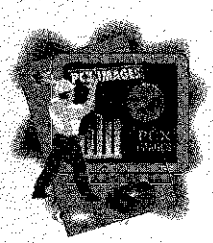
Brian E. Hoffman
Montclair, New Jersey

HAL HARDENBERGH RESPONDS: *Like Mr. Hoffman, I am not going to present a complete rebuttal. He seems to think that I believe that the Japanese are a "sly, imperialistic" threat. Sly? No. Imperialistic? It's the U.S., not Japan, that goes around invading sovereign nations these days. A threat? Anybody who knows about the U.S.-Japan trade imbalance knows Japan is an economic threat.*

In Japan, the statement, "I'm from the government, I'm here to help you" is accepted at face value. In this country, it's a very bad joke. The best Japanese engineers do not work on $860 million bombers which have no military mission. Japanese parents seem to strike a better balance between TV and homework for their children than their U.S. counterparts. Japan has roughly 80 times fewer lawyers per unit population than the U.S. For these and other reasons, I have great respect for Japan and for the Japanese. If Mr. Hoffman thinks I was Japan-bashing, I suggest he read the column again.

The Sherman Anti-trust Act was enacted a century ago. Business conditions have changed since then, especially with respect to foreign competition. I think the time has come to revise that Act. So do some members of the U.S. Commerce Department and Congress. Serious discussions have been held this

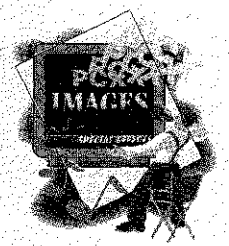
ADD THE POWER OF GRAPHICS TO YOUR PROGRAMS WITH GENUS



PCX PROGRAMMER'S TOOLKIT 4.0

Finally, a programmer's toolkit that helps you incorporate graphics into your programs quickly and easily. The PCX Programmer's Toolkit includes over 75 routines to display, save, scale, and print PCX bitmapped graphics from almost any program.

ONLY $249



PCX EFFECTS*

Add spectacular special effects (F/X) to your programs, without being trapped in someone else's editor or slideshow program. Wipe, split, crush, slide, sand, drip, diagonal, spiral, random, or explode your graphics.

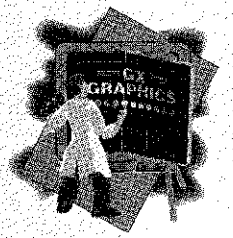
ONLY $99

* Requires PCX Toolkit

PCX TEXT

Now you can display blazing bitmapped text in any graphics mode as simply as text mode. With PCX Text, you can display your textual information right along with your graphical data.

ONLY $149



GX GRAPHICS

A complete graphics library supporting all graphics primitives. Use the GX Graphics toolkit instead of the BGI or MS libraries and make your program faster, smaller, and more portable across compilers – while supporting more video modes.

ONLY $149

FOR THE GENUS GX DEVELOPMENT SERIES CATALOG, CALL TOLL-FREE:

1-800-227-0918

VISA, MASTERCARD, AMERICAN EXPRESS, CHECK, C.O.D., PURCHASE ORDER

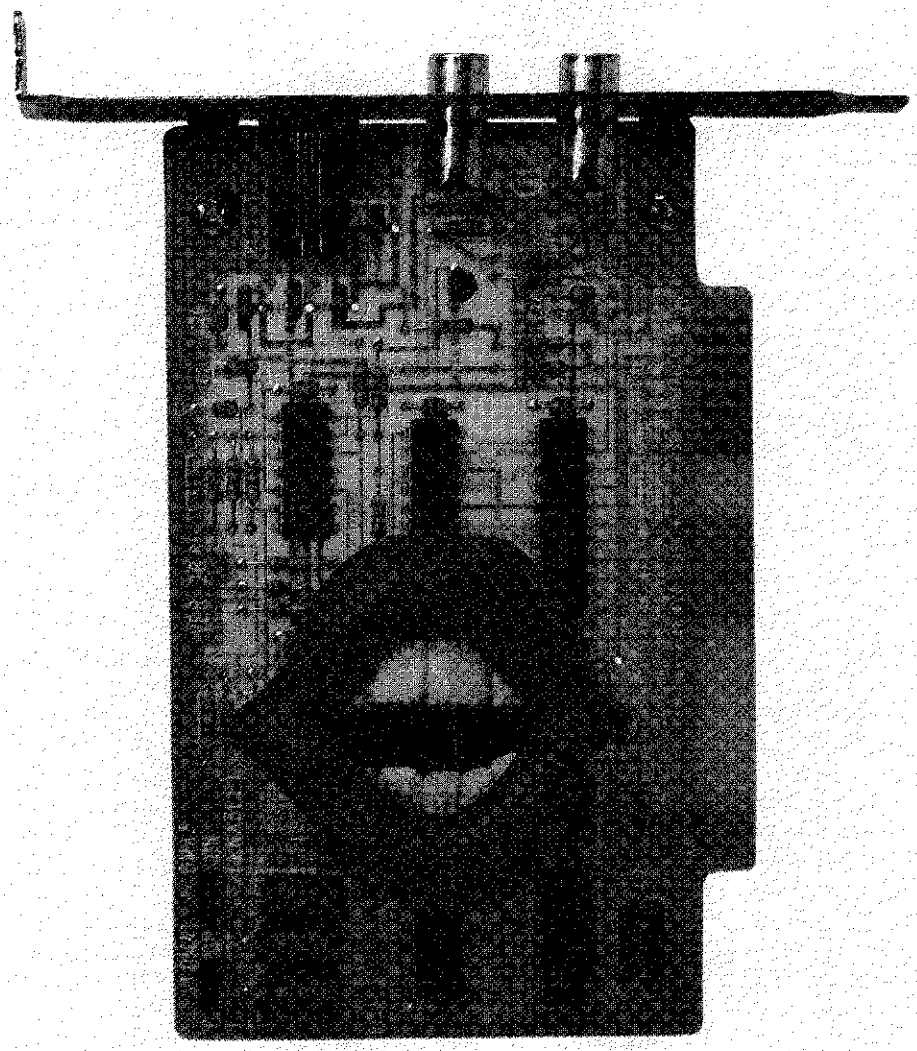
G·E·N·U·S
MICROPROGRAMMING

11315 MEADOW LAKE • HOUSTON, TX 77077 • (713) 870-0737
FAX (713) 870-0288 • BBS (713) 266-9362

Circle 261 on Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Make your software laugh, sob, growl, warble, wail, sing and talk.



If you've ever wanted to add sound to your software, put on your headphones. The LANtastic™ Voice Programmers Interface (VPI) and the LANtastic Voice Adapter are waiting.

Now you can create educational and training programs on disk or CD-ROM that actually speak to students.

Or voice mail packages that send speech across a LAN.

Or "talking" software demos.

Music synthesis.

Sound effects.

And whatever else echoes through your imagination.

You see, the LANtastic Voice Adapter is a low-cost expansion device that converts between audio signals and a digital data stream. It has mono tape in and out jacks so you can record from a CD player, tape recorder, microphone or other external sound source, then play back through an amplifier with speakers, or through the telephone handset provided with each adapter.

Better yet, the LANtastic VPI is a set of simple, straight-forward software tools, which make it possible for your applications programs to use the LANtastic Voice card to record and play digitized sound. And no one else offers anything like it. Especially at $149* per adapter and $195 for the VPI.

No wonder *PC Magazine* (July 1990) wrote: "Leonardo DaVinci might have called it a work of art." *And*: "LANtastic's strong VPI makes it a natural choice for developers wishing to add another dimension to their communications."

For complete information about Artisoft's Voice Developers Kit, call 602-293-6363.

ARTISOFT

*Manufacturer's suggested retail price. ©1990 ARTISOFT. LANtastic is a trademark of Artisoft. Brand names and product names are trademarks or registered trademarks of their respective companies.

See us at COMDEX/Fall '90

November 12–16, 1990
Las Vegas, Nevada
Booth #N2482

Circle 118 on Reader Service Card

PRODUCED BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

year in D.C. to consider modifying the Act to permit industry consortia along the MITI model.

I agree that actions have consequences. In fact, I began the column in question by pointing out that you cannot support non-workers as though they were productive workers (as Denmark and Sweden do) without massively taxing those who do work.

I asserted that Japan concentrated on production quality. Mr. Hoffman states that Japanese companies concentrated on manufacturing. *We seem to be having a violent agreement!*

I respectfully disagree with Mr. Hoffman's assertion that the Japanese non-taxation of interest on savings accounts is not a major factor in the huge difference in capital costs between Japan and the U.S.

FORTRAN FOREVER

Thank you for an excellent overview of 386 FORTRAN compilers (*PJ* 8.4, p. 53). I must tell you that the illustration which accompanies the article bothers me a bit. I believe that I speak for many in the FORTRAN community when I say that I am tired of constantly being told that my language of choice is obsolete, primitive, and "Neanderthal."

A skilled programmer using FORTRAN and assembly languages can do *anything* other programmers can do in the currently-trendy language (I believe that C is the "in" language right now).

If we could measure the "popularity" of languages worldwide, I strongly suspect that we would discover that FORTRAN has:

- More total lines of source code devoted to it than does any other language (except, possibly, COBOL);
- More megabytes of disk storage devoted to it (source, object, and I/O files) than does any other language;
- More machine cycles devoted to executing it than any other language.

Thus, it's hardly an "obsolete" or backwater language!

I once proposed a theorem, which I will now reveal to the world: On any planet where a high-level language is invented, the very first one which is of any use whatsoever will become immortal, and will outlive civilizations, races, and species. On this planet, that language is FORTRAN. Corollary: If the human race destroys itself, its evolutionary successors will be programming in FORTRAN.

The reason for this is obvious: the human race has simply made too great an investment in FORTRAN code to abandon the language. Languages may come and go, but FORTRAN alone is for the ages.

James F. Glass
Chatsworth, California

VIRTUAL MEMORY BUGS

I have some comments to make on the article, "A virtual memory system in C/C++" in *PJ* 8.5. First of all, the C++ code contains several errors. On page 47: "First, a copy constructor is supplied, but copying one virtual array to another is an expensive operation. *The existing destination array must be discarded,* a new one created, and it must be initialized one element at a time. Use this constructor with care."

The advice to "use this constructor with care" is certainly apt, if it does what it says. *What* existing destination array? The author is seriously mistaken about what a constructor does and when it is used. At the top of the second column on page 50 appears:

```
if (hdr && !vfree(hdr) {
```

This statement, which triggers discarding the "existing array," will cause problems since **hdr** is as yet unintialized. It will contain garbage and will usually result in an attempt to **vfree** a garbage value. Since the code is loaded with error traps, this may not have been tested.

I noticed other problems, too. For example, **operator[ ]** contains:

```
if (!(cur_ele = vread( hdr, index )) )
```

which should generate a compiler error since **vread** returns a **void*** and **cur_ele** is an **int***.

Besides all this, I don't like the approach to how **operator[ ]** works. By selecting an index and then accessing, the programmer prevents general use within an expression. If an operator is built like that, it ought to work in the general case.

The simple solution is to use an auxiliary class. All of the overloaded operators that are listed are completely unnecessary, since one could simply have **operator[ ]** return an instance of a class which had **operator=( )** and **operator int( )** defined. While this is mentioned in passing, the writer may have had a different idea in mind.

Any operators which modify the value could also be added, such as ++ and +=, but I found that this was not really worthwhile. Most of the time, I'm dealing with large aggregate objects anyway, not simply with **int**s.

Speaking of **operator++( )**, the author is mistaken in the statement, "as is required by a deficiency in C++, postfix and prefix are not distinguished. They are always treated as prefix expressions, even if the ++ or -- follows the operand." That is an obsolete problem, given the newer compilers. With the older ones, don't try using ++ as a postfix when only the prefix version has been defined, for the sake of compatibility.

The author has a valid point in not liking the overhead of temporary variables in the use of an auxiliary class. A programmer could use **operator[ ]** for ease of expressiveness, and use explicit **get** and **put** functions for efficiency.

I think he missed the boat again, though, on making the **access** class general-purpose. He states that such a class for other types could easily be made, but that it is done by duplicating the entire class. He should have made a general-purpose class that then had a very small wrapper for specific types.

John M. Dlugosz
Plano, Texas

ALLEN HOLUB RESPONDS. *Mr. Dlugosz has made several points in his letter, which I'll address in turn. First, my thanks for finding the bug in the copy constructor. A copy constructor is used by the compiler to copy into uninitialized memory (as compared to* **operator=( )***, which copies into initialized memory). Consequently, the entire* **if** *clause at the top of the right column on page 50 should indeed be deleted.*

I guess I lucked out and got zeroed memory when I tested the routines. I stand by my comments about using the copy constructor with care, however. For example, you shouldn't pass a **vint** *to a subroutine by value, return a* **vint***, and so forth—use a pointer or reference. I suppose the problem could be solved*

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Language

AI Expert Systems
ARITY Combination Pkg	974
Exsys Professional	694
PC Scheme LISP by TI	74
Turbo Prolog Toolbox	70
TURBO PROLOG V2.0	110
VP Expert	219

Assemblers
Advantage Disassembler	279
ASM Flow	89
MS Macro Asm	97
OPTASM	129
Re:Source	129
Turbo Assembler/Debugger	98
Visible Computer 80286	84

Basic & Addons
DB/Lib	120
MS Basic 7.0	339
MS QuickBASIC V4.5	69
Probas by Hammerly	call
ProScreen by Hammerly	call
QBase	124
QuickPak Professional	139
Softcode	69
True Basic	69
Turbo Basic	69

C LanguageCompilers
High C 386	875
Lattice CV6	189
Microsoft QuickC w/QASM	135
MSC 6.0	345
Top Speed C Std	179
Turbo Cby Borland	97
Turbo C Professional	165
Watcom C8.0 386 std.	719

Cobol
MS Cobol V3.0	629
Microfocus COBOL W/tools	1499
Realia COBOL	844

Fortran
For_C FORTRAN'77 to C	670
Lahey FORTRAN F77L	525
Lahey Personal FORTRAN	84
MS FORTRAN with Codeview	295
RM Fortran	494

Modula2
Logitech Dev. Sys	195
Topspeed 3 pak	159

Object Oriented Programming
ACTOR	559
Whitewater Resource Tool Kit	160
Smalltalk/V 286	169
Turbo C++	109
Turbo C++ Prof	159
Turbo Debugger & Tools	99
Zortech C++ with source	205
Zortech Dev't Kit	360

Other Languages
IntegrAda	724
Janus Ada	369
Meridian Ada Dev't Kit	980
MuLISP87	219
PC/Forth+	199
Personal Rexx	129

Turbo Pascal
Turbo ASYNCH PLUS	115
Turbo Pascal 5.0	105
Turbo Pascal 5.0 Professional	175
Turbo POWER TOOLS PLUS	124
Turbo Programmer	444

DBMS

Data Base Development
DBXL	199
Clarion Prof	525
Clipper S87	399
dBASE III Plus	450
Force	395
FoxBASE +	239
Clipper 5.0	495
Fox Pro Lan	745
Fox Pro SU	579
QuickSilver	379

dBASE Library
Artful	189
CodeBASE IV	215
Funky	175
IDL	125
Netlib	229
Proclip V3.5	119

File Management
Btrieve	180
Cindex Plus	329
CISAM	325
CQL wSRC	349
WPASS	349
ctree	call
dBC III PLUS	435
db_FILE RETRIEVE	265
Essential BTree w/source	149
FABS Plus	172
Netware C interf.	239
Netware SQL	489
pBase	185
Turbo Programmer/C	449
WKS Lib	189
XDBC	555
XQL	599
Xtrieve PLUS	459
Btree Filer	97

Addons

C Communications
Newl C Async Lib. sy Silverware	209
BreakOut II	99
C ASYNCH MANAGER	155
Essential Communications	295
Greenleaf Comm Library	279
Greenleaf ViewComm	315
Lattice Comm. Library	179

Dbase Addons
AdComm 99	105
BLINK	179
CLEAR + FOR dBASE	139
Dbase Online V2.0	125
dBugsource debugger	170
dBXdBASE III to C	460
dClip	189
DCS Report Gen	265
desQ Clipper Api	469
dGEbusiness graphics	259
Documentor	225
dQuery	165
dSalvage	79
FLIPPER Graphics Library	175
Spellcode	89
Touch & Go	325
R&R Code Generator	125
R&R Relational Reportwriter	125
Scrimagescreen/menu	129
SilverComm Library V2	169
Silver Paint Library	79
Steve Straly's Tools	169
UI ProgrammerDev's Version 2.0	419

General Addons
C Tools PlusV5.0	105
C Utilitiesby Essential	154
Greenleaf Functions	159
Greenleaf SuperFunctions	209

Graphic Addons
Essential Graphics	345
Graphic C	329
GraphicsMenu	159
Greenleaf MakeformDOS	90
GSS Dev't Toolkit	505
Halo '88140 + dvices	275
Menuet	259
MetaWINDOW/PLUS	269
Mewel	149
PCX Programmer's Toolkit	175
Slate w/Graphics	415

Text Screen Addons
AEWindows	204
C Worthy w/forms	259
Cursesby Aspen Scientific	125
Greenleaf DataWindows	325
HiScreen XL	124
HAM by JYACC	464
VC Screenpainter	114
Vitamin Csource, menus	164

Utilities
BACKIT	120
COPY II PC	35
Disk Technician Advanced	149
FASTBACK Plus	142
MACE GOLD	129
MACE Utilities	90
Norton Utilities Advanced	101
PC/Tools Delixe 6.0	95
V opt	111
XENOCOPYPC	70
XTree Pro	111

Miscellaneous

Other Products
Aspen Korn Shell	109
CO/SESSION	214
DESQ 386	call
dQuery/Lib	274
Line & Locate ++-ROM MSC.	339
Printware	149
Smartem 240	294
Stingray Clarion Lems	call
QEMM 386	55
386 Maxpro	115

Case & Prototypes
Demo II	155
NEW! Easy Case +	205
Instant ReplayNostradamus	134
Interactive EasyFlow	135
MetaDesign by Meta Software	329
ProC	425
PROTEUS Prototype System	124
Show Partner F/Xdemos	315
Softdemo	59
Protoview	499

Tools

Debuggers
386 DEBUG	155
MultiScope	135
Periscope OK	465
Softprobe 86/PX	345

Development Tools
DASM	239
MKS AWK	84
MKS Lex & Yacc	205
MKS RCS	155
PC Lint	call
Plink 86 Plus overlays	399
PolyMake	129
PVCS Corporate	355
.RTLINK + by Pocket Soft	409
TLib	89

Editors
BRIEF	call
Epsilon	155
KEDIT	119
Me wsource	169
MKS Vi	125
MultiEdit	90

★ SAGE PROFESSIONAL EDITOR ★
Mouse Support, DOS/OS/2,Emulates Brief, vi & Emacs, EMS Support Fully ProgrammableList 295 Ours 249
30day Money Back Guarantee

NROFF/PC	117
PC/EDIT+	269
Qedit	79
SPF/PC	189

Windows & OS/2

Windows
Asymetrix Toolbook	call
Case:W	754
CTalk/views	375
Dbfast/Windows	325
DialogCoder	429
MS Windows SDK	325
MultiScope Windows	285
Object Graphics	319
ProtoView	619
RF Flow	69
WinTrieve	335
WNDX Gui Tools	445
Windows 3.0	99

OS/2
Brief	call
Btrieve	475
C Tree	309
C Worthy	339
Greenleaf Data Windows	319
Mks Awk	159
Mks Lex & Yacc	339
Mks Toolkit	399
Multiscope	340
OS/2 PM Toolkit	359
VC Screen	139
Vitamin C	289
PC Yacc	620
Zortech OS/2	129

1-800-323-1809

PROGRAMMERS WAREHOUSE

7819 East Greenway Suite 8 • Scottsdale, AZ 85260 • Outside US call (602) 443-0580

All Prices and Offers Subject to Change Without Notice

PRODUCED BY PCMCIA.ORG
ELECTRONIC REPRODUCTION PROHIBITED

by removing the copy constructor entirely, but then you couldn't do this:

```
vint x( 100L );    // code that initializes x
vint y = x;
```

Also, Mr. Dlugosz is right in saying that the compiler should have rejected the ***cur_ele = vread(...)*** *assignment.*

Ellis and Stroustrup's Annotated C++ Reference *claims that newer compilers require an explicit cast to go to and from* ***void****. (C doesn't require the cast.) The problem is solved easily enough by casting* ***vread*** *to* ***int****. Zortech C 2.0 accepted the code as it stands. Turbo C++ (which I didn't have running when I wrote the article) correctly prints an error.*

His comments about ***operator++( )*** *being an "obsolete problem" are completely out of line. Neither Zortech nor Turbo C++ distinguish between prefix and postfix overloads. Again, Ellis and Stroustrup discuss a new feature of the language that allows the compiler to distinguish between the two operators. (****operator++****(void) defines a prefix operation and* ***operator++****(int) defines a postfix. The argument is just a dummy—0 is passed into the function at run time.) I'd like to take his advice and "get a newer compiler." Can he suggest a commercially available MS-DOS C++ compiler that supports this feature?*

Mr. Dlugosz is also entitled to his opinion about the general organization of the ***vint*** *class, but I think* he's *missed the boat when he suggests a* ***get/put*** *strategy. I really wanted to be able to have a statement like this:*

```
int i;    vint a(100L), b(100L);    ...
a[i] = b[i];
```

and have it be reasonably efficient. A ***get/put*** *system such as the one he suggests would require something like:*

```
a.put(i, b.get(i) );
```

which to my mind is pretty unreadable.

The whole point of using C++ is to get away from this sort of messy interface, and I see little point in sacrificing readability for efficiency when you can have both.

*Finally, Mr. Dlugosz's comments about not needing the various operator overloads (+, -, *, and /) are misguided in the general case, though on reflection I agree with him in the current situation—it's true that you can get by with only the conversion and assignment operators. This strategy is probably appropriate here, because virtual-array members are* ***ints****, so the conversion overhead is very low. There's often a high overhead associated with conversion, however, and that overhead isn't present in an operator overload.*

Anyway, my intention was to demonstrate how to use the C virtual-memory functions in a general way in C++, selecting an ***int*** *class because it was easy to understand. It seemed reasonable to demonstrate a good generalized overall structure that was easily adapted to other situations.*

MISOSYS, Inc.

Programmer's Journal downloads are now available from MISOSYS' forum on CompuServe: PCS49

When you don't have to write in stone, don't let your editor weigh you down. You need SAID-86!

Editing was never so easy!

SAID-86 is a fast, flexible, full screen text editor for PC's. It is perfect for editing batch files, program listings, README files, CONFIG.SYS files, and anything you now do with EDLIN or the non-document mode of a word processor. Why struggle with huge editors; when all is said and done, SAID-86 will be your text editor of choice!

Check out this list of features

- ✓ WordStar-like editing commands are easy to use
- ✓ Pull-down menu system for commanding SAID-86
- ✓ Supports nine editing buffers with automatic swap to disk
- ✓ Supports up to 30 user-defined macros; 255 characters each
- ✓ Undelete the last nine deleted lines can save your bacon
- ✓ MOUSE support with automatic recognition
- ✓ HELP facility; shell to invoke DOS commands from SAID-86
- ✓ SAID-86 can expand or contract TABs

SAID-86 is reasonably priced at just $29.95 + $2S&H

Now get the clearest view of your disk drive and memory contents with our DED86™

Powerful features in Version 2!

When you need to travel through your disk drive, why settle for a tool that isolates you from the road? DED86 gives you the direct controls you need to explore your disk. It's a full-screen sector-oriented disk/file editor and a page-oriented memory editor. When you want to "unerase" erased files, DED86's flexible KEEP facility does the job without you fussing over FATs.

Check out this list of features

- ✓ Look by cylinder/head/sector, sector or cluster; even reserved/hidden sectors
- ✓ Scan free clusters to search for erased data ✓ Jump about subdirectories
- ✓ Keep sectors & clusters for writing to a file; Great for partial file recovery
- ✓ Edit bytes in hexadecimal or ASCII, zap in 0s; Undo last edit!
- ✓ Search your disk drive, a file, or memory for ASCII or hexadecimal strings
- ✓ Obtain complete disk statistics in one screen; includes usage data
- ✓ Save/Restore sectors to/from auxiliary buffers; for moving data around
- ✓ Handles 5.25" & 3.5" drives & RAM disks; even large partition drives

DED86 is easy on your pocketbook; only $29.95 + $3S&H

IIT Math Coprocessors

IIT's CMOS coprocessors use less power, execute faster, support 4 x 4 matrix transformation, have thirty-two 80-bit numeric registers, come with a factory 5-year warranty, and cost less! If you use your machine for spreadsheets, desktop publishing, CAD, etc., consider an IIT coprocessor. S&H is $5 in U.S.

IT-2C87-100	$237.00
IT-2C87-125	$289.00
IT-2C87-200	$307.00
IT-3C87-20	$349.00
IT-3C87-25	$447.00
IT-3C87-33	$545.00
IT-3C87SX-16	$354.00
IT-3C87SX-20	$385.00

MISOSYS, Inc.
P.O. Box 239
Sterling, VA 22170-0239
800-MISOSYS
(US&Canada)
or 703-450-4181
M/C & VISA accepted.
S&H are U.S. only

EXPANZ!™ Disk Expander Card

With the new EXPANZ! data compression card, you can boost hard disk capacity up to three times. EXPANZ! plugs into any open slot and intercepts calls to and from the disk controllers. Compresses and decompresses in real time. Requires PC/XT/AT or compatible running DOS 3.x. **Just $179.10 + $6S&H.**

Circle 121 on Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Witness Software Development History.

In 1911, the greatest scientists in the world came to Brussels for The First Solvay Congress. It was the first time that Einstein, Curie, Planck, and Poincare could be heard in one place at one time. When it ended, a new era in modern science began.

In 1991, the most prominent software developers in the world will descend upon Silicon Valley for Software Development '91. They will reveal new ways to achieve dramatic gains in software development productivity. When it's over, the next era of application development will begin.

The best and brightest programmers in the country want to show you how to write cleaner, faster, more elegant code —*at Software Development '91. In* just three-and-a-half days, they will transform the way your work by revealing new tricks, techniques, and technologies you can use as soon as you get back to work. You'll put more fun into your programming while you improve application quality.

Why You Should Attend

You re-energize your programming activity, because you learn more ways to eliminate mundane tasks and increase your creative power.

This is the most efficient way to stay on top of the sweeping changes taking place in software development.

No other single event gives you access to the advice and counsel of so many industry leaders.

Tour the Exhibits to Witness Hot New Products in Action

Try out and compare newly released products designed to help you develop your applications. Put products to the test with hands-on demonstrations. Talk to knowledgeable vendors to find out how products are changing, what's hot, and what to look for in the months ahead.

Authoritative Sponsorship

Software Development '91 is sponsored by Miller Freeman Publications, publishers of *Computer Language, UNIX Review, AI Expert, Database Programming and Design, Embedded Systems Programming, LAN Magazine,* and *Software Development International.*

You'll find the high quality you've come to expect from these magazines at Software Development '91.

It's History in the Making! Join over 2,000 of your colleagues from around the world, who know that the best way to prepare for the future in software development is to be there as it's being made. They'll be meeting again at Software Development '91 — the fourth annual conference by and for software developers.

- It's all happening February 12-15, 1991 at the Santa Clara Convention Center, Santa Clara, CA
- They'll be there to review and test the hundreds of new development techniques in over 200 lectures and workshop sessions. And they'll be previewing the latest programming tools at the products exhibition — in over 150 booths

You can and should be there — return this card for complete information.

SOFTWARE DEVELOPMENT '91

❑ I'd like to take a look at Software Development History in the making— Send me complete information on Software Development '91

❑ I'm also interested in exhibiting.

Name _____

Title _____

Company _____

Street _____

City, State, Zip _____

Phone/FAX _____

You'll also receive Software Development Seminars' quarterly newsletter — FREE.

Send to: Software Development '91, 500 Howard Street, San Francisco, CA 94105

PJ11

Send Today For More Information

Call: (415)995-2472

FAX: (415)267-7683

or write
Software Development Seminars
500 Howard Street
San Francisco, California
94105

Circle 167 on Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Redirections

By Frank D. Greco

Look for Sun to introduce their SPARCstation II (and expect a name change) any day now. The new workstation appears to be a 30 MIPS (40 MHz clock) workstation priced at the current SPARCstation 1+ price. That means that we'll soon be seeing a fully-equipped high-performance UNIX-based color workstation with approximately 300 megabytes of disk storage for somewhere between $10,000 and $15,000. Apparently the only decision still to be made is whether to drop the price of the current SPARCstation or just eliminate it entirely.

I would wager that Sun will drop the current SPARCstation 1+ product. My guess is that they will just go with their color IPC and monochrome SLC models at the low end and with the new SPARCstation at the high end of their workstation line.

MORE SPARC STUFF

Keep a look out for an 80 MIPS superscalar version of the SPARC processor, a joint venture from Hyundai Electronics, LSI Logic, and Metaflow Technologies, to be available by mid-1991. This chip employs a technique known as "speculative execution" which the processor can use to avoid waiting at unresolved conditional branches. It does this by continuing instruction fetches along their most probable path. The engineers of this joint venture are predicting 300 MIPS and above when they build a multiprocessor version of this workstation in 1993.

Sun's only independent reseller, Interactive Systems Corporation, has disclosed agreements to port Sun's version of UNIX, SunOS, to SPARC machines from several UNIX hardware vendors. Rumor has it that the companies are KFW Inc., Able Technologies, Twinhead Corp, TriGem Computer, Fujitsu, and Star Technologies. All of these vendors are scheduled to announce and ship SPARC clones in 1991.

LITTLE BIG MAN

Quantum Corporation has announced the availability of 425-megabyte, 3.5-inch SCSI drives in evaluation quantities to qualified resellers. I would guess that by mid-1991, these speedy (12-14 ms) drives will be initially priced at around $1,500. During the next few years, most UNIX workstations will be offering drives of this capacity as standard equipment.

FUI ON MOTIF GUI

Those of you who think Motif either is or will be the standard Graphical User Interface (GUI) for workstations are wrong, in my humble opinion. Sun Microsystems is strongly behind the Open Look standard which they, along with AT&T and Xerox, have championed for several years. Sun will not—I repeat, will not—formally support OSF's Motif GUI.

In a short period of time, there will be a plethora of high-powered, low-priced SPARC-based workstations from Sun and other vendors running System V.4 in corporate America. Guess what the standard user interface for System V.4 is? You guessed it—Open Look. It doesn't matter what OSF mandates, because it's the market that ultimately dictates the standard. Despite the constant repeating of history, some people just never learn.

Remember a few years ago when the ANSI group was settling on a standard for BASIC? Well, how many versions of ANSI BASIC did we see on IBM PCs? It was Microsoft BASIC, and **not** ANSI BASIC, which ended up as the standard for that language.

I have personally evaluated both GUIs for several corporate clients, and all my clients are opting for Open Look and **not** Motif. Wall Street firms are overwhelmingly choosing Open Look as their corporate GUI. As opposed to what OSF wants you to think (using some very strong mud-slinging tactics), there are only a very few who are using or developing for Motif.

MFLOPPY DRIVES

Intel has been quietly telling the customers of its hot 80 MFLOPS 860 chip that a 300-400 MFLOPS version will be available in mid-1991. The chip's designers are also planning for the future 860 to handle context switches much more efficiently than the 860 does now, so vendors won't be in the position of having to design a 960 to help it out.

GETTING CARDED

Canon has joined the optical card business. These credit-card-sized optical storage devices hold two megabytes of laser-read data and are perfect for transporting graphics, audio, or multimedia electronic documents. Faithful "Redirections" readers will remember that when I first mentioned these inexpensive storage devices a few years ago, a ten-megabyte version was in the works by another vendor. According to some informed sources, the Canon announcement is imminent.

DOMO ARIGATO AUSPEX

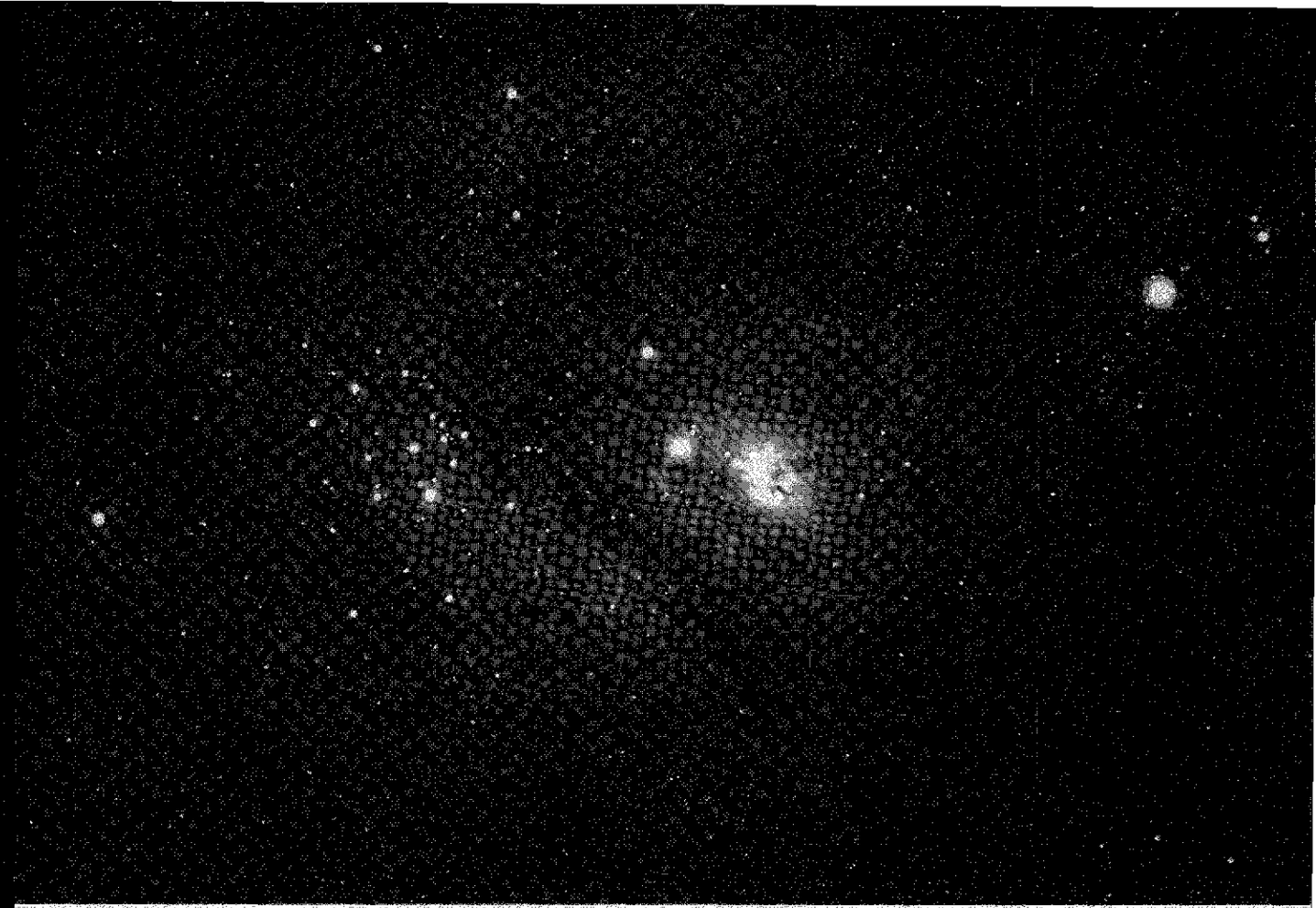
Fuji Xerox and Nissho Electronics have both invested millions in Auspex Systems, a vendor of a high-performance NFS file server. As part of the investment, each company will receive a 15% ownership of the growing Auspex.

MORE SAX AND VIOLINS

Being a musician myself, I have a special interest in the Musical Instrument Digital Interface (MIDI). In spite of its popularity in the recording industry, there are many deficiencies with the current MIDI standard. For one thing, it's way too slow. It can only handle 16 channels at a tepid 31.25K bits per second. In addition, the number of MIDI devices that you can put on a bus is extremely limited.

Frank D. Greco is an independent consultant who provides industry analysis, systems consulting and technological forecasting for Fortune 500 companies. Interested readers may get in contact with Frank c/o Crossroads Technologies, P.O. Box 530, Fanwood, New Jersey 07023, (201) 754-7820.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED



Lagoon Nebula, a 50-light-year-wide cloud of dust and gas, 4,500 light-years from Earth. National Optical Astronomy Observatories, Kitt Peak National Observatory. Resource: "The Universe and Beyond," Terence Dickinson, Camden House Publishing.

Travel the Universe Beyond 640K

.RTLink®/Plus 4.0 with Virtual Memory Linking

Programs execute in as little as 100K and/or take advantage of all available memory.

The .RTLink/Plus linker provides memory management tools which allow MS-DOS programs larger than 640K to execute. However, .RTLink/Plus is not just for large programs. When LAN requirements, Multitaskers, TSRs, etc., squeeze your program out of conventional memory, .RTLink/Plus allows execution in the space available.

.RTLink/Plus provides a complete set of Memory Management/Linker tools: VML™ (Virtual Memory Linking); Multiple and Nested Overlay capability; and VML combined with overlays. VML will get your program running without overlays or other hassles. If fine tuning is required for your particular application, .RTLink/Plus allows simple or detailed adjustments to be made.

.RTLink/Plus automatically takes advantage of the target machine's memory environment. Programs will execute in conventional or expanded memory (if present). Virtual 'pages' or overlays will automatically cache intelligently using conventional memory or disk...or from expanded or extended memory, if present.

.RTLink/Plus provides a software solution for DOS memory problems which is valid for XT, AT, 386 and above.

No source code changes are required.

A high quality, built-in CPU profiler is provided at no additional charge.

For additional information, including the list of currently supported programming languages

WRITE: Pocket Soft, Inc.
4.0 Info Pack
P.O. Box 821049
Houston, TX 77282
USA

FAX: 713-460-2651
CALL: 713-460-5600

Pocket Soft, Inc.

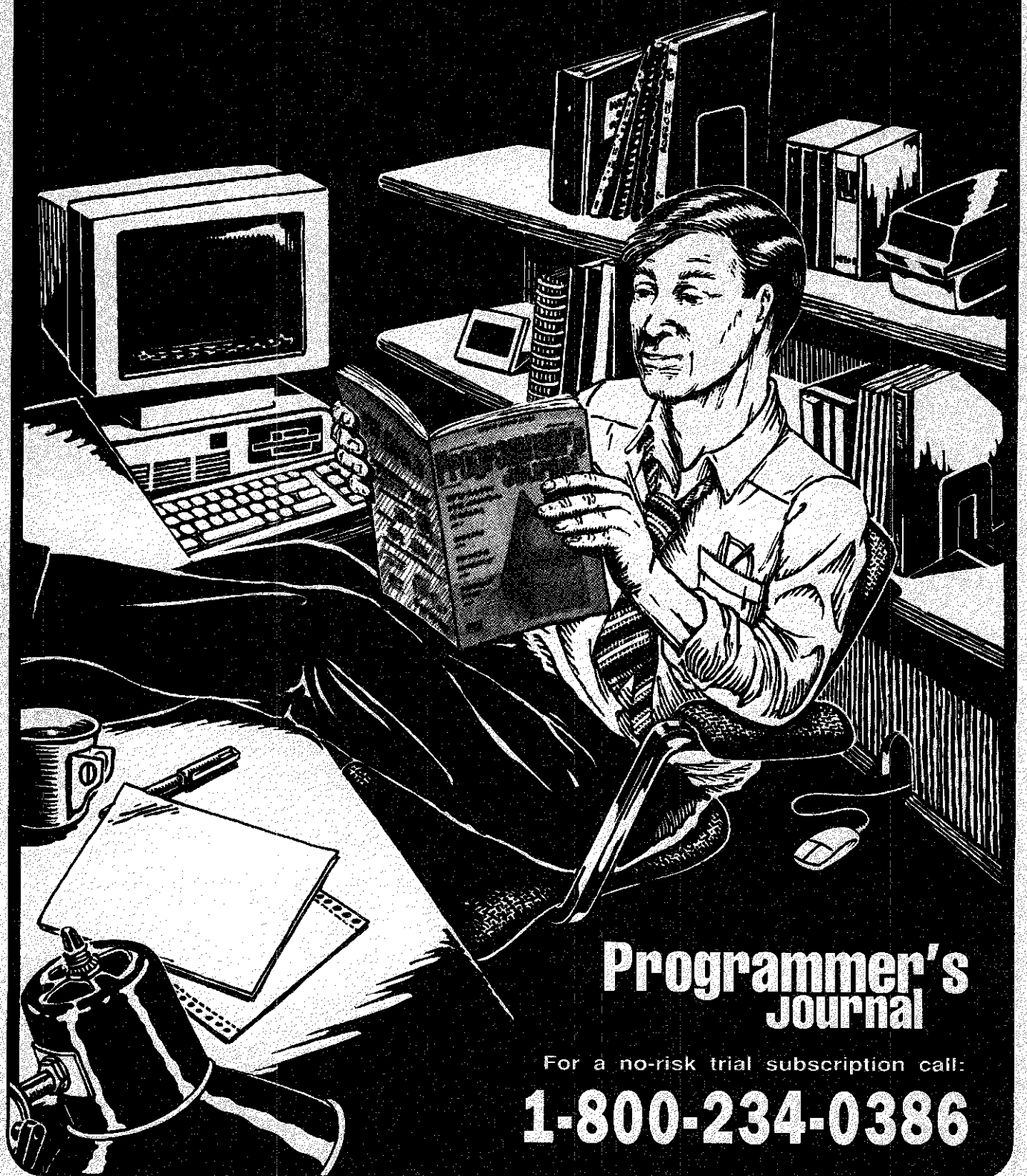
P.O. Box 821049 • Houston, Texas 77282

RTLink is a registered trademark of Pocket Soft, Inc. VML and Pocket Soft are trademarks of Pocket Soft, Inc.

Circle 279 on Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

README.



Programmer's Journal

For a no-risk trial subscription call:

1-800-234-0386

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

LISTING 1

```
/* To test out my version of system() that handles csh/ksh -isms:
 *  cc -o pgm main.c basename.c system.c      */

#include <stdio.h>

static char *_desc = "Copyright 1990 Frank D. Greco";

main(argc,argv)
char **argv;
{
    if (argc != 1)
        system(argv[1]);
}
```

LISTING 2

```
/* basename.c - Returns last component of pathname.
 *    Included here in case your system doesn't offer it   */

#include <stdio.h>
#include <signal.h>
#include <sys/fcntl.h>

#define TTY         "/dev/tty"
#define DIR_DELIM   '/'

static char *_desc = "Copyright 1990 Frank D. Greco";

char * basename(cp)
register char *cp;
{
    register char *ptr = cp + strlen(cp) - 1;

    while(*ptr != DIR_DELIM  ||  ptr < cp )
        ptr--;
    return ptr + 1;
}

/* system.c -   System that uses shell indicated by SHELL env variable
 *       instead of Bourne shell.                    */

system(cp)
char *cp;
{
    int status;         /* Status of spawned process */
    int pid;            /* Process ID of spawned process */
    int waitstatus;         /* Return code from wait() */
    int tty;            /* File Descriptor of logical terminal */
    int (*intstat)();       /* Fn ptrs for signal() */
    int (*quitstat)();
    char *shell, *shellname;  /* Pathname and Name of shell */
    extern char *getenv();
    extern char *basename();  /* Fn to get last component of pathname */

    fflush(stdout);
    if ( (tty = open(TTY, O_RDWR)) == -1 ) {
        fprintf(stderr, "Cannot open %s\n", TTY);
        return -1;
    }
    if ( !(pid = fork()) )  {       /* Create a subprocess */
        close(0);   dup(tty);   /* Adjust file descriptors */
        close(1);   dup(tty);
        close(2);   dup(tty);
        close(tty);
        if ( !(shell = getenv("SHELL")) )   /* Get shell from env */
            shell = "/bin/sh";

        shellname = basename(shell);        /* Get tail end of path */
        execlp(shell, shellname, "-c", cp, (char *) 0);
        exit(127);                          /* Can't normally get here */
    }
    close(tty);
    intstat = (int (*)())signal(SIGINT, SIG_IGN);   /* Ignore SIGINT */
    quitstat = (int (*)())signal(SIGQUIT, SIG_IGN); /* Ignore SIGQUIT */

    /* Now wait for spawned process to terminate */
    while( (waitstatus = wait(&status)) != pid  &&  waitstatus != -1 )
        ;
    if ( waitstatus == -1 )
        status = -1;
    signal(SIGINT, intstat);            /* Restore signal handlers */
    signal(SIGQUIT, quitstat);
    return status;
}
```


OPUS SOFTWARE

"Without a doubt, OPUS Make is the hottest make utility on the market"
Tom Swan, PCWorld

OPUS Make

For professional programmers, OPUS Make is the superior choice in a make utility. It is the fastest most full-featured make utility there is. Features include:

- DOS version uses only 9K memory!
- Multiple directory support.
- Supports: Polytron PVCS™, Burton Systems TLIB™, Microsoft LIB™, SLR Systems OPTLIB™
- Generates automatic response files of unlimited length for LINK and LIB.
- OPUS MKMF included: An automatic dependency generator that fully understands C preprocessor directives.

Both DOS and OS/2 executables for only: $129 MC, Visa, COD, or PO, 60 day MBG!

Call 1-800-248-OPUS
International: 415-594-7501
1032 Irving Street, Suite 439, San Francisco, CA 94122
Trademarks owned by their respective companies

Circle 254 on Reader Service Card

The Programmer's Corner
— A Source Code Repository —

If you are reading this magazine, then you appreciate the art of programming. One of the most important tools for a software craftsman is example code. To meet this need, we provide **more than 9000 files** (over one gigabyte) of shareware and public domain source code and programs. We cover all popular languages including C, Pascal, and Basic. TPC even has hard to find source code such as OS/2 PM C source, C++ source, and Turbo Pascal 5.5 OOP source. Source code from this issue of *Programmer's Journal* is also available. New code is added daily, all in the interest of helping the professional programmer.

So call today: (301) 596-1180 300-9600 Baud
(301) 997-3744

Only $50 for a one year subscription

— TPC uses FastComm 9600 V.32 Modems —

TPC

The Programmer's Corner
5430 F Lynx Lane, Suite 224
Columbia, MD 21044

Circle 54 on Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

So what's the answer? Use LAN technology. That's what vendor Lone Wolf, Inc. has done with their MediaLink protocol. MediaLink is MIDI-compatible, digital/audio compatible, SCSI compatible, and HDTV compatible. You could handle a maximum of 524,288 devices on MediaLink at speeds around two megabits per second, and eventually you would approach FDDI speed (100 Mbps). MediaLink converts MIDI data into network packets that contain addressing and MIDI device information, and it can handle point-to-point, broadcasting, or multicasting. Many musical instrument manufacturers are agreeing that this architecture is the way to go for the next generation of MIDI.

ALL THE NEWS FIT TO PRINT

Because it's more than a little unhappy with the high royalty fees that it must pay Adobe for the use of Postscript in a laser printer, Sun should soon introduce their own low-cost laser printer. Similar to the strategy taken by Steve Jobs and the engineers at NeXT, Sun's printer will essentially handle bits. A SPARC-based workstation connected to the printer will use Sun's marvelous Postscript superset (and window system), NeWS, to render the image in the workstation's RAM memory. It will then blast the bits out from a fiber-optic connection to the printer. Expect this printer to cost about $1,000.

I SEE THE LIGHT

Micracor, Inc. has been licensed to manufacture solid-state lasers which are based on technology from MIT's Lincoln Laboratories. These microchip lasers will be useful for super-high-speed optical computers. The effective "cycle time" of these chips is somewhere in the neighborhood of 80 picoseconds, about a 1000-fold increase over the current non-optical switching times. What's more, the chairman of Micracor indicates that these microchip lasers will be available in quantity for 10 cents. Yipes!

FINDING A PEARL IN BLUEPOINT

Adobe Systems has quietly acquired the technological assets of BluePoint Technologies, a maker of a proprietary chip set that renders fonts from algorithmic outline descriptions. Adobe apparently has plans to use BluePoint chips in future high-speed versions of their Postscript controllers.

A SOURCE OF A DIFFERENT COLOR

Most of us are already familiar with the **system** library call available in a UNIX environment. **System** is a higher-level library call than an operating system call is. It spawns a process to execute the shell command which has been specified as its argument. Although the **system** call is fairly slow, it is effective. Versions of this command have even migrated to the DOS world.

HARVARD SOFTWORKS

NUMBER ONE IN FORTH INNOVATION

(513) 748-0390 P.O. Box 69, Springboro, OH 45066

MEET THAT DEADLINE ! ! !

- Combine the raw power of extensible languages with the convenience of carefully implemented functions!
- Execute faster than optimized C!
- Compile 40,000 lines per minute!
- Stay totally interactive, even while compiling!
- Program at any level of abstraction from machine code thru application specific language with equal ease and efficiency!
- Modify routines without recompiling!
- Use source code for 2500 functions!
- Use data structures, control structures, and interface protocols from any other language!
- Implement that borrowed feature, often more efficiently than in its source!
- Use an architecture that supports very small programs or full megabyte ones with a single version!
- Forget chaotic syntax requirements!
- Use subroutine libraries written for other languages! More efficiently!
- Outproduce experienced programmers stuck using conventional languages!

HS/FORTH with FOOPS - The only full multiple inheritance object oriented language under MSDOS!

WAKE UP ! ! !

Forth is no longer a language that tempts programmers with "great expectations", then frustrates them with the need to reinvent simple tools expected in any commercial language.

HS/FORTH Meets Your Needs!

We don't shortchange you with promises. We provide implemented functions to help you complete your application quickly.

Don't judge Forth by public domain products or ones from vendors primarily interested in consulting - they profit from not providing needed tools! Public domain versions are cheap - if your time is worthless. Useful only in learning Forth's basics, they fail at showing its true potential. Not to mention being s-l-o-w.

You can't add extensibility to fossilized compilers. You can easily add features from any other language to **HS/FORTH**. Try our full modifiable multiple inheritance **OOPs** extension or our **Fortran** translator & mathpak!

HS/FORTH has more features and libraries than we can possibly describe here. **Please ask for our brochure!**

HS/FORTH uses MSDOS or PCDOS. Each level includes features of lower ones. Level upgrades: $25. plus price difference. Sources in ordinary text files.

STUDENT LEVEL **$145.**
text & scaled/clipped graphic windows, ellipses, splines, arcs, fills, turtles; powerful parsing, formatting, file and device I/O; shells; interrupt handlers

PERSONAL LEVEL **$245.**
software floating point, trig, transcendental, quad & scaled integer, automatic optimizer for assembly like performance.

PROFESSIONAL LEVEL **$395.**
hardware floating point, turnkey, seal, Rosetta Stone Dynamic Linker, round robin & time slice multitaskers, dynamic string manager, file blocks, sector mapped blocks, assemblers

PRODUCTION LEVEL **$495.**
Metacompiler: DOS/ROM/direct/indirect C data structures & struct compiler, TurboWindow-C graphics library,

Options:

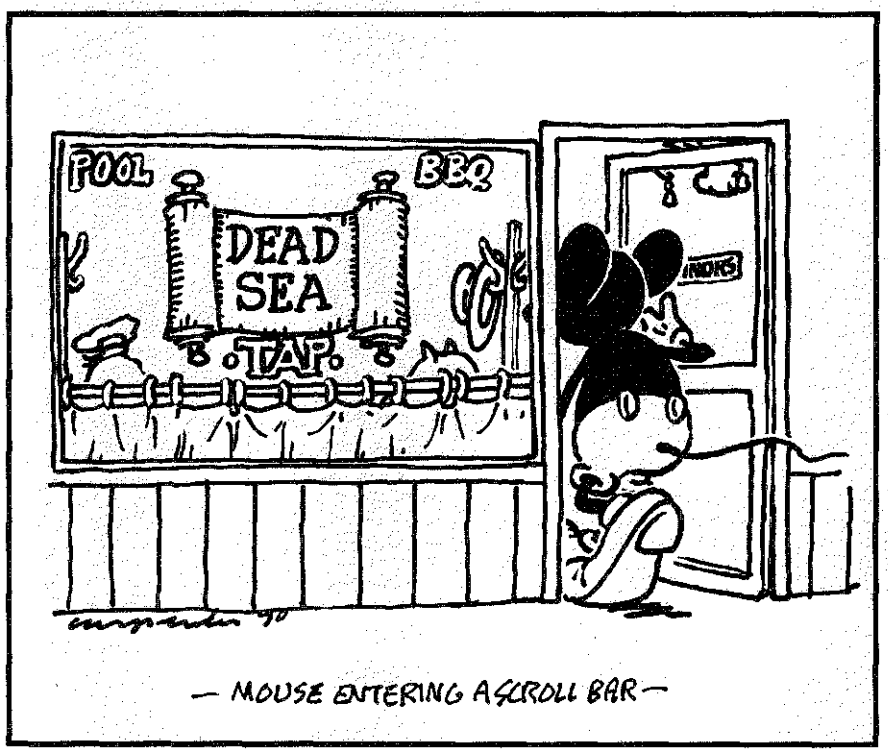
FOOPS+ with multiple inheritance	**$ 75.**
286FORTH or **386FORTH**	**$295.**
TurboWindow-C (Metagraphics)	**$ 99.**
BTRIEVE for HS/FORTH (Novell)	**$199.**
ROMULUS HS/FORTH from ROM	**$ 95.**
FFORTRAN translator/mathpak	**$ 75.**

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

One small but annoying "problem" with the UNIX version, though, is that it's hard-coded to use the Bourne shell (**sh**) for running the requested command. Sometimes, especially if you're a fan of Bill Joy's C Shell (**csh**), you might want to use a **csh**-ism in your line. The same thing applies if you're a fan of the newer Korn shell. Unfortunately, only the Bourne shell syntax can be used in the argument to **system**.

The code that accompanies this article is my own version of **system**. It searches for an environment variable called "SHELL" and spawns whatever shell was specified. This variable should be set to the full pathname of the shell you want to use. Also, since I've used a library call, **basename**, in my version of **system** which some of you might not have available on your UNIX system, I've included a function that accomplishes the same thing. Sample code for this is given in Listing 2.

Have fun. PJ



World's Only Compressing Linker!

SLR SYSTEMS is **YOUR** source for the fastest, most powerful linking technology available. Wherever you are headed, chances are we are there, or already working on it.

OPTLINK technology is designed to have no inherent limitations. There is NO internal limit on the number of modules, libraries, publics, segments, etc. The only limit is the amount of disk space available.

OPTLINK/Compress

- **Compression yields quick self-extract-to-run .EXE files up to 50% smaller than /EXEPACK.**
- **Storage of larger program on one floppy disk.**
- **Quick self-extract code is efficient for most applications.**
- **File needs less than 1K extra memory for decompression.**
- ***Only $250***

"We've been using various versions of OPTLINK for years ... the compression works wonderfully."
Dave Nanian
UNDERWARE, INC.
Original Author of **BRIEF**

"A breakthrough product! Saved us from adding another disk on several of our products."
Alex Young
Project Manager
INTUIT

"Compress uniquely gives me small code quickly, which is great!"
David Grawrock
CENTRAL POINT SOFTWARE

SLR SYSTEMS INC.

1622 N. Main Street, Butler, PA 16001 (412) 282-0864 Voice (412) 282-7965 FAX (412) 282-2799 BBS

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

More VGA color complexity

VGA color paging

By Michael Abrash

Installment 21: *In which we carry on with VGA color.*

When last we left our hero, he/she (take your pick) had learned how to set the VGA's Digital to Analog Converter (DAC) to select 16 or 256 colors from a set of 256K colors. However, there was still a great deal to learn about color paging, loading and reading the DAC, and color cycling, so that's the direction we're headed next. Before we begin, though, let me clear up a potential problem with C code from earlier installments.

BROKEN GRAPHICS CODE

Microsoft C 6.0 may be the greatest C implementation ever, but it broke much of the C graphics code I've presented in this column. You see, in order to set a single pixel in EGA/VGA memory, it's necessary to perform a read to latch display memory, followed by a write to draw the pixel. (If you don't understand why this is the case, I suggest that you look back at the first few installments of this column. For now, just take my word for it.) Ideally, we'd like to perform the read and write in close succession, with a single instruction if possible. In assembly language, that's no problem: just **xchg**, **and**, **xor**, or **or** an appropriate value with memory.

In the code in question, the value written to memory is irrelevant because of the way set/reset and the bit mask are set up; all that's needed is a read followed by a write with any value at all. Matters are not so simple in C. Long ago, I tried to use code like ***ScreenPtr |= 0;*** to perform a read/write operation to display memory.

Unfortunately, the optimizer in Microsoft C 5.0 (not 6.0) considered this to be a null operation, because any value **or**ed with 0 remains unchanged (when the destination is system memory, that is; as described above, **or**ing with 0 is a useful operation when performed to VGA memory). Having concluded that it had encountered what amounted to a null operation, MSC 5.0 then declined to generate any code.

That was easy enough to fix. I just replaced it with ***ScreenPtr |= 0xFF;***, which is definitely not a null operation. (Remember, for our purposes the value **or**ed with display memory is irrelevant, so 0xFF served just as well as 0.) MSC 5.0 obligingly assembled the desired **or** with memory, and there matters stood until the arrival of MSC 6.0. MSC 6.0 doesn't think ***ScreenPtr |= 0xFF;*** is a null operation, but it doesn't think it's an **or** operation either. 0xFF is the invariable result from **or**ing a value with 0xFF, so MSC 6.0 treated that code as if it were ***ScreenPtr = 0xFF;*** and assembled a **mov** instruction, not an **or** instruction. That eliminated the read we needed before writing to display memory, and that's what broke so much of the graphics code.

SOLVING THE PROBLEM

The solution is simple. When using C code from past installments, replace any line like ***ScreenPtr |= 0xFF;*** with ***ScreenPtr |= 0xFE;*** and you should be all set. This is necessary for much of the C code published in this column since Day 1, if you're using Microsoft C 6.0. (This fix is implemented in Listing 1.)

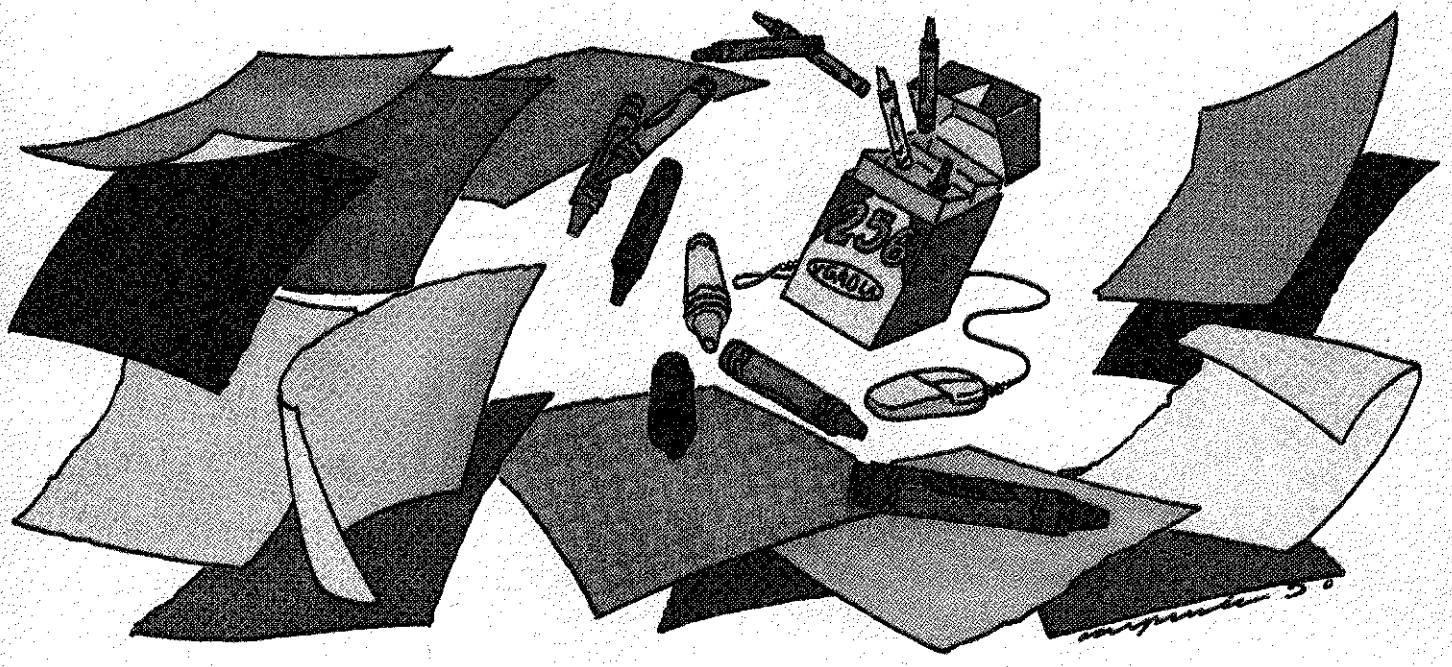
There's an important point to all this beyond keeping you from sending the editors nasty letters saying my code doesn't work. Many people believe that it's possible to write low-level graphics code in C that's as efficient as code written in assembly language, and more portable, if you know exactly what code the compiler generates.

This example shows why that's a fool's game; code that depends on the compiler's code generation isn't necessarily even portable from one release of the compiler to the next, let alone between compilers. Besides, the graphics adapters you're likely to encounter under MS-DOS are generally found

---

Contributing Editor Michael Abrash works for Metagraphics of Scotts Valley, California as a senior engineer. He is the author of Power Graphics Programming *and* Zen of Assembly Language. *Readers may write to Michael at P.O. Box 414, Burlington, VT 05402-0414, or contact him via MCI Mail, user name* MABRASH.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED



only in 80x86-based computers, so it's hard to imagine what portability benefits there might be. On top of that, I'm still waiting to see C graphics code that matches good assembly language code in the performance department.

All in all, my advice is to stick to assembly language for your key graphics code. The foregoing notwithstanding, I'm going to use C exclusively in this installment because C is more compact, easier to read, and perfectly satisfactory in this application; knowing when it's not worth the trouble of optimizing is an important part of being an effective programmer. With that out of the way, let's move on to color paging.

A 16-COLOR ARRANGEMENT

As you'll recall from the last installment, the VGA's DAC contains 256 storage locations, each holding one 18-bit color organized as one 6-bit red component, one 6-bit green component, and one 6-bit blue component. In 256-color mode, each 8-bit pixel value in memory selects one of the 256 DAC locations. The value in that location is then looked up and sent to the screen as the pixel's color, and that's that. In 16-color modes, however, each pixel is represented in display memory by a 4-bit value, so each pixel can only look up one of 16 DAC locations to be sent to the screen.

But what about those other 240 DAC locations? Are they all useless in 16-color modes? Not at all. Normally (as configured by the BIOS at mode set time in 16-color modes), the first 64 locations in the DAC are set to colors that are equivalent to the 64 colors an EGA can display. (The other 192 DAC locations may or may not be set to any particular values, so be sure to load them before relying on their contents.) At the same time, the EGA-compatible palette RAM is set to the same values an EGA is normally set to. This is dandy if you're writing code for an EGA, because when you tweak the palette RAM on a VGA you'll get exactly the results you'd expect on an EGA. If you're writing VGA-specific code, though, you're shortchanging yourself.

The ideal 16-color arrangement on a VGA is as follows. Load palette RAM register 0 with the value 0, palette RAM register 1 with the value 1, and so on up to palette RAM register 15, which should be set to 15. (I described how to load the palette RAM in the last installment, and in a little while we'll see working code that loads the palette.) The object here is to cause the palette RAM to pass pixel values through unchanged, so we can ignore it; the DAC will do all the color work.

Now load DAC locations 0 through 15 with any 16 colors you'd care to display. A pixel value of 0 in memory will cause the color in DAC location 0 to be displayed, a pixel value of 1 in memory will select DAC location 1, and so on. You can change the values stored in the DAC whenever you want to work with a different set of colors. (Loading the DAC was covered last time, and will be touched on again shortly.)

COLOR PAGING

Things get more exciting with the addition of color paging. If you set bit 7 of the Attribute Controller Mode register (Attribute Controller register 10h) to 1, then bits 3 through 0 of the Color Select register (Attribute Controller register 14h) become bits 7 through 4 of each 8-bit value used to address locations within the DAC. The Color Select register operates in conjunction with the pixel data from display memory, which provides DAC address bits 3 through 0 (see Figure 1). I'll call this mode of operation "16-pages-of-16-colors" paging.

When bit 7 of the AC Mode register is 0 (the EGA-compatible default), "4-pages-of-64-colors" paging is selected, as described in the last installment.

The Color Select register gives us the ability to perform color paging, that is, to flip instantly between color sets and thereby provide a new color interpretation for every pixel on the screen without changing the contents of display memory. If you view the DAC as consisting of 16 pages containing 16 colors each, as shown in Figure 2, then the Color Select register selects one of those pages and the pixel data chooses one of 16 colors from within that page. (Remember that the palette RAM is set

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

CAD ENGINE

SilverScreen Licensing Program

SilverScreen is a 3D CAD and Solid Modeling software system that contains a resident C Language compiler. The Ccompiler was designed by Schroff Development for use by end users and software developers to facilitate the development of custom applications requiring 3D and solid modeling.

SilverScreen is the most powerful and versatile 3D CAD/Solid Modeling system available containing such features as: 2D and 3D Boolean operations; rendering; shading; mass properties; text editor; associative dimensioning (ANSI Standard); object-oriented database; schema-based attributes; and camera walk. SilverScreen runs on DOS and UNIX and supports a wide range of mice, video boards, printers, plotters and digitizers. File transfer is handled through DXF, IGES, HPGL, PostScript and others. Silverscreen features direct export to SmartCAM and Stereolithography (3D Systems).

SilverScreen's resident C compiler implements a library of more than 300 functions plus a large number of functions dedicated to accessing the SilverScreen database (from which parts lists and bills of materials can be generated) and to controlling the SilverScreen environment. Custom menus and icons also may be created. SilverScreen also adds the powerful ability to embed the SilverScreen script commands in a program for direct execution by SilverScreen.

For SilverScreen information contact:
Schroff Development Corporation
P.O. Box 1334
Mission, KS 66222
(913) 262-2664
For Software Developer's information contact:
OEM Department
(913) 722-2826
Evaluation packages available

See *SilverScreen* at these shows:

C-Forum '90 October 17 - 19 **Boston**

SYSTEC '90 October 22 - 26 **Munich, Germany**

AUTOFACT November 13 - 15 **Detroit**

SilverScreen®

3D CAD/Solid Modeling Software

Circle 65 on Reader Service Card

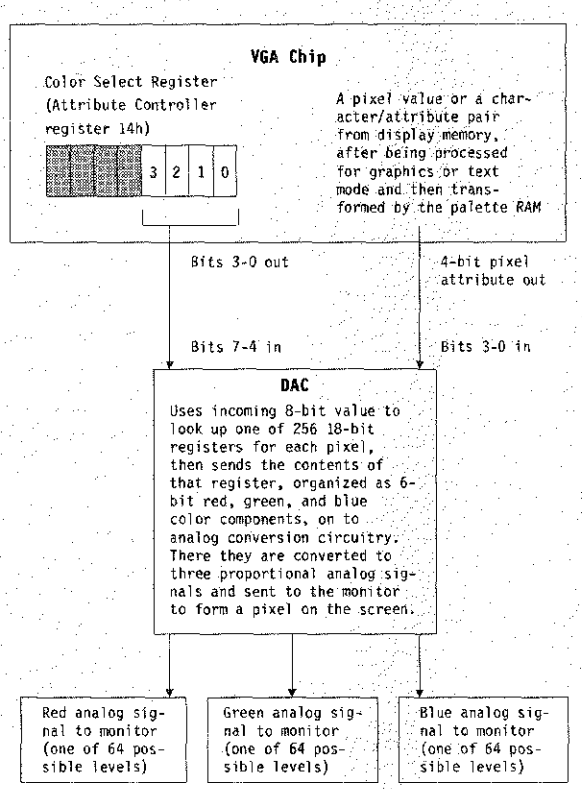
LISTING 1

```
/* Illustrates color paging by color-animating a series of
 * concentric circles to produce the illusion of motion.
 * Runs on the VGA only, because color paging isn't available on
 * the EGA.
 */

#include <dos.h>

#define USE_BIOS  1  /* set to 1 to use BIOS functions to perform
                        color paging, 0 to program the color paging
                        registers directly */

/* Handle differences between Turbo C & MSC. Note that Turbo C accepts
   outp as a synonym for outportb, but not outpw for outport */
#ifdef __TURBOC__
#define outpw  outport
#endif

#define SCREEN_WIDTH_IN_BYTES 80     /* # of bytes across one scan
                                        line in mode 12h */
#define SCREEN_SEGMENT        0xA000 /* mode 12h display memory seg */
#define GC_INDEX              0x3CE  /* Graphics Controller index */
#define SET_RESET_INDEX       0      /* Set/Reset reg index in GC */
#define SET_RESET_ENABLE_INDEX 1     /* Set/Reset Enable reg index
                                        in GC */
#define BIT_MASK_INDEX        8      /* Bit Mask reg index in GC */
#define INPUT_STATUS_1        0x3DA  /* Input Status 1 port */
#define AC_INDEX              0x3C0  /* Attribute Controller index */
#define AC_DATA_W             0x3C0  /* Attribute Controller data
                                        register for writes */
#define AC_DATA_R             0x3C1  /* Attribute Controller data
                                        register for reads */
#define AC_MODE_INDEX         0x30   /* AC Mode reg index, with bit 6
                                        set to avoid blanking screen */
#define AC_COLOR_SELECT_INDEX 0x34   /* Color Select reg index, with
                                        bit 6 set to avoid blanking
                                        screen */

int main();
void DrawDot(int X, int Y);
void DrawCircle(int X, int Y, int Radius, int Color);

/* Array used to load the DAC. Organized as 256 RGB triplets */
static unsigned char DACSettings[256*3];

/* Array used to set the palette RAM to a pass-through state.
   The 17th entry sets the border color to 0 */
static unsigned char PaletteRAMSettings[] = {
   0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 0};

main() {
   int Radius, Color, Page, Element, i;
   union REGS Regs;
   struct SREGS Sregs;
   unsigned char GreenComponent;

   /* Select VGA's hi-res 640x480 graphics mode, mode 12h */
   Regs.x.ax = 0x0012;
   int86(0x10, &Regs, &Regs);

   /* Draw concentric circles */
   for ( Radius = 10, Color = 1; Radius < 240; Radius += 2 ) {
      DrawCircle(640/2, 480/2, Radius, Color);
      if (++Color >= 16)
         Color = 1;                 /* skip color 0 */
   }

   /* Load the upper 240 DAC locations (15 pages) with one-position
      rotations of a series of increasingly green colors. Because
      page 0 is being displayed, the screen remains unchanged while
      the other 15 color pages are being loaded */

   /* First, fill DACSettings with the desired green settings
      (because it's a static array, all locations are initialized to
      zero, so we don't need to initialize the red or green color
      components, which we want to be zero) */
   for ( Page = 1; Page <= 15; Page++ ) {
      GreenComponent = Page * 4;
      for ( Element = 1; Element <= 15; Element++ ) {
         DACSettings[Page*16*3 + Element*3 + 1] = GreenComponent;
         if ( (GreenComponent += 4) >= 64 )
            GreenComponent = 4;
      }
```

continued

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

FIGURE 1


VGA Chip
Color Select Register
(Attribute Controller
register 14h)
3 2 1 0
A pixel value or a character/attribute pair from display memory, after being processed for graphics or text mode and then transformed by the palette RAM
Bits 3-0 out
4-bit pixel attribute out
Bits 7-4 in
Bits 3-0 in
DAC
Uses incoming 8-bit value to look up one of 256 18-bit registers for each pixel, then sends the contents of that register, organized as 6-bit red, green, and blue color components, on to analog conversion circuitry. There they are converted to three proportional analog signals and sent to the monitor to form a pixel on the screen.
Red analog signal to monitor (one of 64 possible levels)
Green analog signal to monitor (one of 64 possible levels)
Blue analog signal to monitor (one of 64 possible levels)

to a pass-through state, so pixel values of 0 through 15 come through the palette RAM to the DAC unaltered.)

Basically, the Color Select register gives you instantaneous access to any one of 16 completely independent pages of 16 colors each. Big deal, you say; why not just reprogram the first 16 DAC locations if you want different colors? First, it's considerably faster to program the Color Select register with a few I/O operations than it is to reprogram 16 DAC registers, a process which takes 64 **outs**. Second, it's much easier to avoid flicker when using the Color Select register; all you have to do is wait for the vertical sync pulse and set one register. In fact, you don't even have to wait for the vertical sync pulse because, to the best of my knowledge, the colors will change instantly, with no glitching, starting at the location of the electron beam at the very instant you set the Color Select register. (If you happen to know better, please correct me.)

The only reason to wait for the sync pulse is to make sure that all the pixels in each frame are drawn with the same color set, in order to avoid having the top and bottom of the screen briefly appear to be mismatched.

LOADING THE DAC

On the other hand, loading the DAC without flicker or glitching is no picnic. The loading has to take place during non-display time; otherwise, for a short time part of the screen will be drawn with a mix of the old color set and the new color set, resulting in unintended and often highly undesirable on-screen effects. However, it can be difficult on slow computers to load a large block of DAC locations during a single vertical blanking period, so there may be no way to reload the DAC cleanly between one frame and the next, especially on an XT (although this is more of a problem in 256-color mode, where it's sometimes necessary to reload all 256 DAC locations simultaneously). Furthermore, some BIOSes glitch or bounce the screen when called to load the DAC, and some are faster than others.

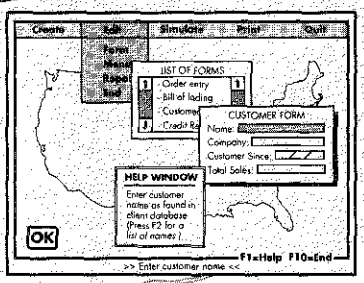
We'll return to issues of loading the DAC in the next installment. For now,

HI-SCREEN XL™

THE PROVEN USER-INTERFACE SOLUTION FOR YOUR PROGRAMS

Since 1983, Softway has specialized in user-interface development tools aimed at demanding professional programmers.

WINDOWS • MENUS DATA ENTRY • HELP



HI-SCREEN XL offers a high-level modular approach to screen management that provides for superior programming **control, flexibility** and **speed.**

- The built-in **screen editor** is used to create screens/windows/menus along with all field descriptions. Any **Hercules, CGA, EGA** or **VGA** graphics can also be captured. Screens are then saved in a compressed file format outside your program.
- **High-level calls** from your program give you total control over the user interface. Combine text and graphics, dialog boxes, context-sensitive help, elaborate data entry & menu management, keyboard & mouse control, etc.

WITH MOUSE AND GRAPHICS SUPPORT

- Supports all versions of **C, Basic, Cobol, Fortran, Pascal, dBase, Assembly,** etc.
- Use linkable (with C and MS Basic) or removable TSR screen driver (30-40K).

HI-SCREEN XL **$149**
HI-SCREEN XL *Professional Series*
(with mouse & graphics support): **$325**

30 day money-back guarantee • No royalties

Call today: **1-800-338-2852**
Tel.: 415-896-0708 • Fax: 415-896-0709

SOFTWAY, Inc.
185 Berry Street, Suite 5411, San Francisco, CA 94107

Circle 110 on Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

INSTALL.BAT

First impressions last.

INSTALL 3.0 gives your software product a professional introduction.

Installations that rely on batch files look amateurish and have limited error handling. INSTALL 3.0 creates an elegant installation procedure that will increase user's confidence in your software product.

INSTALL 3.0 comes with many samples you can modify and use immediately; no programming usually needed. All available RAM is used for fast file transfers.

- Features very sophisticated error-handling
- Does high performance data compression
- Can safely and reliably modify system files
- Allows selective installation of parts of your program
- Complete source code included
- No royalty charges
- Free technical support

INSTALL 3.0 detects monitors, DOS versions, coprocessors, etc. to create machine-specific installations without asking questions.

INSTALL 3.0 has been used for over four years by some of the biggest (and smallest!) names in the industry, in the U.S. and abroad, to install millions of copies of programs. Make sure your software product makes a good first impression with INSTALL 3.0.

KNOWLEDGE DYNAMICS CORP.
Highway Contract 4, Box 185-H
Canyon Lake, TX 78133-3508
MasterCard/VISA/COD/POs welcome

$249.95 + shipping
Money back guarantee
Sales 800-331-2783
Internat'l 512-964-3994
24 hr FAX 512-964-3958
24 hr BBS 512-964-3929

Circle **295** on Reader Service Card

Data Junction® converts data files!

Any ASCII • dBASE • R:BASE • Lotus 1-2-3 Excel • Paradox • Informix • Q&A • Oracle • Xdb askSam • SC4 • Clarion • DIF • SYLK • Symphony ACT! • Platinum • DAC • and other application files IBM/Mips/Sparc/Cray/Vax/Intel/68000 Binaries Btrieve • c-tree • EBCDIC • Packed data
. . . and more!

- *No programming-completely menu driven!*
- *Select and rearrange records, fields and characters!*
- *Instant script file creation for total batch processing!*
- *Map new data structures automatically by name!*
- *Ideal for OEM applications!*

". . . an invaluable integration tool." *-Byte*
". . . more than worth its price." *-PC Magazine*
". . . a surefire winner." *-Computer Language*

Standard $99 - dBASE, ASCII, 1-2-3, Q&A, Enable, DIF, askSam & merge files for WordPerfect, Word & WordStar

Professional $199 - Standard formats plus Binary/EBCDIC, SC4, SYLK Paradox, R:BASE, Clarion, Excel, c-tree & Btrieve

Advanced $299 - Standard & Professional formats plus Informix, Xdb, Oracle & C-ISAM

Call **800/444-1945** or 512/482-0824
MC/VISA • AMEX • COD • Corp. PO.
available for LANs, Xenix & Unix
Tools & Techniques Inc. • 1620 W 12 • Austin TX 78703

T&T TOOLS & TECHNIQUES

Circle **327** on Reader Service Card


```
   }

   /* Now call the BIOS to load the upper 240 DAC locations */
   Regs.h.ah = 0x10; /* INT 0x10 color control function # */
   Regs.h.al = 0x12; /* INT 0x10 load DAC block subfunction # */
   Regs.x.bx = 16;   /* DAC location #16 is the first to load */
   Regs.x.cx = 240;  /* load 240 DAC locations */
   Regs.x.dx = (unsigned int)(DACSettings + 16*3);
   segread(&Sregs);
   Sregs.es = Sregs.ds;       /* point ES:DX to DACSettings */
   int86x(0x10, &Regs, &Regs, &Sregs);

   /* Put the palette RAM in a pass-through state and set the Overscan
      register (border color) to 0. We've saved this for last because
      it changes the colors being displayed */
   Regs.h.ah = 0x10; /* INT 0x10 color control function # */
   Regs.h.al = 2;    /* INT 0x10 set palette RAM block subfunction # */
   Regs.x.dx = (unsigned int)PaletteRAMSettings;
   segread(&Sregs);
   Sregs.es = Sregs.ds;       /* point ES:DX to PaletteRAMSettings */
   int86x(0x10, &Regs, &Regs, &Sregs);

   /* Enable 16-pages-of-16-colors paging */
#if USE_BIOS
   Regs.h.ah = 0x10; /* INT 0x10 color control function # */
   Regs.h.al = 0x13; /* INT 0x10 color paging control subfunction # */
   Regs.h.bl = 0;    /* INT 0x10 color paging type sub-subfunction # */
   Regs.h.bh = 1;    /* select 16-pages-of-16-colors paging */
   int86(0x10, &Regs, &Regs);
#else
   inp(INPUT_STATUS_1);                /* reset AC index/data toggle */
   outp(AC_INDEX, AC_MODE_INDEX);      /* point to the AC Mode reg */
   outp(AC_DATA_W, inp(AC_DATA_R) | 0x80); /* select 16-pages paging */
#endif /* USE_BIOS */

   /* We're ready to roll; the upper 15 color pages are set up, the
      palette RAM is in a pass-through state, and 16-pages-of-16-
      colors paging is enabled. Now we'll loop through and display
      for one frame each pages 15 through 1 and then back to 15
      for a while */
   for ( Page = 15, i = 0 ; i < 1000; i++ ) {

#if USE_BIOS
      /* Select the desired color page */
      Regs.h.ah = 0x10; /* INT 0x10 color control function # */
      Regs.h.al = 0x13; /* INT 0x10 color paging control subfunction # */
      Regs.h.bl = 1;    /* INT 0x10 color page select sub-subfunction # */
      Regs.h.bh = Page; /* the desired page # */
      int86(0x10, &Regs, &Regs);
#else
      /* Wait for the leading edge of the vertical sync pulse; this
         ensures that we change color pages during vertical
         non-display time, and that the page flips are evenly spaced
         over time */
      while ( (inp(INPUT_STATUS_1) & 0x08) != 0 )
         ;                    /* wait for non-vertical sync time */
      while ( (inp(INPUT_STATUS_1) & 0x08) == 0 )
         ;                    /* wait for vertical sync time */
      inp(INPUT_STATUS_1);          /* reset AC index/data toggle */
      outp(AC_INDEX, AC_COLOR_SELECT_INDEX); /* point to the Color
                                                Select register */
      outp(AC_DATA_W, Page);  /* select the desired page # */
#endif /* USE_BIOS */


      /* Cycle from page 15 down to page 1, and then back to page 15.
         Avoid page 0 entirely */
      if (--Page == 0)
         Page = 15;
   }

   /* Restore text mode and done */
   Regs.x.ax = 0x0003;
   int86(0x10, &Regs, &Regs);
}

/* Draws a pixel at screen coordinate (X,Y) */
void DrawDot(int X, int Y) {
   unsigned char far *ScreenPtr;

   /* Point to the byte the pixel is in */
#ifdef __TURBOC__
   ScreenPtr = MK_FP(SCREEN_SEGMENT,
      (Y * SCREEN_WIDTH_IN_BYTES) + (X / 8));
#else
```

continued

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

what all this amounts to is that if you need to switch color sets frequently and color paging can do the job you need done, it's far superior to constantly reloading the DAC—and easier, too.

HOW TO DO COLOR PAGING

Color paging can be performed either directly or through the BIOS. As usual, the BIOS route is the better choice if there's no good reason not to use it, but there is one possible drawback that we'll come across shortly. In any case, I'll describe both techniques.

Color paging directly (without the BIOS) is a two-step process: the first step is enabling color paging, and the second step is selecting the desired color page. Enabling color paging is a simple matter of setting bit 7 of the AC Mode register to 1, as follows:

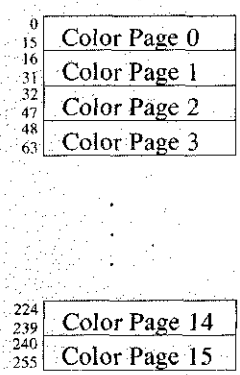
- Read the Input Status 1 register at 3BAh (using monochrome mode) or 3DAh (using color mode) to reset the AC Index/Data toggle;
- Write 30h to the AC Index register at 3C0h to address AC register 10h, the AC Mode register. The index value is 30h rather than 10h because bit 6 of the AC Index register should always be 1, except when setting the palette RAM; when bit 6 is 0, the screen blanks;
- Read the AC Mode register setting from the AC Data register at 3C1h;
- **or** 80h with the value just read to set bit 7 (this selects 16-pages-of-16-colors operation rather than 4-pages-of-64-colors operation);
- Write the result to the AC Mode register via the AC Data register at 3C0h. (Remember that the AC Data register is at 3C1h for reads, but is at 3C0h on every other **out**—alternating with the AC Index register—for writes, in order to provide EGA compatibility.)

At this point, color paging with 16 pages of 16 colors each is enabled and you can select a color page as follows:

- Read the Input Status 1 register at 3XAh to reset the AC Index/Data toggle;
- Write 34h to the AC Index register at 3C0h to address AC register 14h, the Color Select register;
- Write the desired page number (0 through 15) to the Color Select register by way of the AC Data register at 3C0h.

FIGURE 2

When 16-pages-of-16-colors paging is enabled, the 256 DAC locations are logically organized as 16 color pages each containing 16 DAC locations.



Professional software made affordable.

Sandpaper's System Debugger

The power of the 386 is yours for the taking.
System Debugger uses the 386 to create the ideal DOS development environment.
It uses the 386's protection, page tables and debug registers to provide the power of an expensive In-Circuit Emulator at a fraction of the price.

With System Debugger you'll ...
Set breakpoints on memory location and range reads and/or writes and **discover.**
⇒*When is the InDos flag incremented?*
⇒*What subroutine is overwriting my code?*
⇒*When does my malloc() space get corrupted?*
⇒*Am I overwriting DOS?*
Set breakpoints in ROM and **learn.**
⇒*Why does my software work in RAM but not in ROM?*
⇒*How does the BIOS work?*
Set breakpoints on I/O ports and interrupts and **know.**
⇒*Is a TSR conflicting with my software?*

Order now at no risk.
Money-back guarantee.

$99

Here is just a sampling of its features:
- Has symbolic debugging with source code of programs, Device Drivers, TSRs and ISRs.
- Runs standalone as well as in tandem with your current debugger.
- Doesn't slow down the system: load it by autoexec and it's ready when you need it.
- Stays active even if your program is hung with interrupts disabled.
- Displays undocumented DOS structures and memory map.
- Automatically comments interrupt calls so you don't need to look them up.
- Has an easy to learn intuitive windowed user interface.
- Finds those elusive annoying bugs which only exist in real time.
- Runs under DOS on 386SX/386/486 AT compatibles with 256K extended memory.
- Runs entirely in extended memory so that 640K is left where you need it.

To order, Call or FAX (818) 705-2657, or send a check to:

Sandpaper Software

P.O.Box 18935, Encino, CA 91416-8935

Please specify diskette format. CA residents please add sales tax.

Circle 32 on Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Repeat this process whenever you want to switch to another color page.

Although it's not required, you'll generally want to wait for the leading edge of the vertical sync pulse before switching pages; this provides for the smoothest color transition, as described above. Take the vertical sync wait out of Listing 1 and set USE_BIOS to 0, and you'll see that the appearance of the program suffers considerably.

PAGING THROUGH THE BIOS

Controlling color paging through the BIOS is a similar two-step process. The first step is to enable 16-pages-of-16-colors paging by invoking interrupt 10h, the video interrupt, with AH = 10, AL = 13h, BL = 0, and BH = 1. (If BH = 0, 4-pages-of-64-colors operation is selected.) The second step is to select the desired color page by invoking interrupt 10h with AH = 10h, AL = 13h, BL = 1, and BH = the number of the desired page, in the range 0 through 15.

At least one VGA BIOS (the one in my computer) waits for the leading edge of the vertical sync pulse before switching color pages. I don't know if every BIOS does or not, and it's an important question. The problem is that if you wait for the leading edge of vertical sync before calling the BIOS to switch pages, and then the BIOS waits for the leading edge of vertical sync before switching pages, you'll only switch pages once every two frames at most. On the other hand, if you don't wait for vertical sync and neither does the BIOS, you'll end up switching pages in the middle of frames and doing so much too often. As usual, only with your own software do you have complete control and full knowledge of what's going on.

If any of you BIOS experts out there can tell us whether waiting for the leading edge of vertical sync is part of the BIOS spec, and thus reliable, please let us know. For now, you can try it both ways, because Listing 1 supports both BIOS and direct color paging. In Listing 1, I make the assumption that the BIOS does wait.

THE COLOR PAGING STATE

If you want to check whether 16-pages-of-16-colors paging is enabled and which color page is currently selected, you can do that either directly or through the BIOS. Obtaining the color paging state through the BIOS is ridiculously easy. Just invoke INT 10h with AH = 10h and AL = 1Ah. On return, BL will contain the state of bit 7 of the AC Mode register (1 for 16-pages-of-16-colors paging, 0 for 4-pages-of-64-colors), and BH will contain the number of the currently-selected color page.

To obtain the color paging state directly, do the following:

- Read the Input Status 1 register at 3XAh to reset the AC Index/Data toggle;
- Write 30h to the AC Index register at 3C0h in order to address the AC Mode register;

ProGraphx

ProGraphx Menus™

Simply add four lines of code to your program and you can have a pull-down or pull-up menu system. Define a menubar with a text file, compile it and receive events calls back to your program. Change the menubar without recompiling your program to create demo versions or other adaptations. A sample application of a Microcoded processor is included along with its two editors.

ProGraphx Menus™	$99
Menus with Source	199

ProGraphx Toolbox™

Over 50 hi-speed graphics routines for EGA/VGA 16 color modes, including proprietary icon routines to display color images.

ProGraphx Image Editor, a complete color icon paint package to edit and view up to 50 icons at once. Simply paint an icon, then use the *ProGraphx Toolbox™* routines to display it with amazing speed. Over **500** sample icons for use in your own programs. *Operation:Airlift*, a full screen scrolling arcade style game with source code.

All routines written in assembly for fastest possible performance.

ProGraphx ToolBox™	$199
Toolbox with Source	399

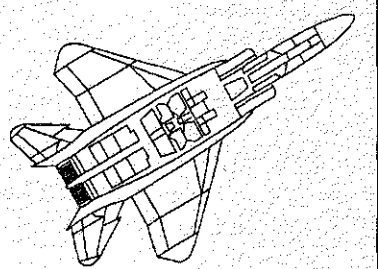
Stanwood Associates, Inc.
303 East Ohio Street Chicago, IL 60611

1-(800) 752-7982

(312) 828-9734 VISA/MC FAX (312) 828-9742

ProGraphx
Speed, Simplicity, and Power for Graphics Programmers

Compatible with Turbo C or Pascal, Zortech C/C++, Microsoft C or any language compatible with 80x86 object code. Uses less than 10K of your code segment!



Also available from :
Programmer's Shop &
Programmer's Connection

Circle 330 on Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

- Read the AC Mode register setting from the AC Data register at 3C1h. Bit 7 of this value indicates the color paging state, as described above;
- Read the Input Status 1 register at 3XAh so as to reset the AC Index/Data toggle;
- Write 34h to the AC Index register at 3C0h to address the Color Select register;
- Read the Color Select register setting from the AC Data register at 3C1h.

AN EXAMPLE OF COLOR PAGING

Listing 1 is an example of color paging. It first sets the display to 640x480 16-color graphics mode, then draws dozens of concentric circles, using code we developed a few installments back. I'm using the C code because speed isn't an issue here and I have limited listing space; refer to *PJ* 8.2, March 1990, for the assembly language code. It's worth noting that Listing 1 is written entirely in C, and that while the drawing part of Listing 1 looks slow, the color paging part does not. By its very nature, which involves working with a few control registers rather than a large bitmap, color manipulation tends to produce snappy results with little effort.

The circles in Listing 1 are drawn in sequential colors 1 through 15, wrapping back from 15 to 1. Color 0 is reserved for the background and is the same in all 16 color pages; otherwise the background would change color as we changed color pages, destroying the subtle color cycling effect we're striving for. The background wouldn't be the only problem, either; the border would also change, a still more bizarre effect.

Remember, the DAC modifies pixels after they've been fully processed by the VGA, and so far as the DAC is concerned pixel attribute 0 is attribute 0 no matter whether it came from the Overscan (border) register, a background pixel with value 0, or by way of translation in the palette RAM. Consequently, you should take care when color paging not to modify the DAC locations that the Overscan and background colors select—unless, of course, you intend to change the border and background colors.

This is a good time to point out that the DAC is **always** available and active. It's obvious that the DAC can be used to select color sets in 256- and 16-color modes, but the DAC can process whatever comes out of the VGA in any mode. You can use the DAC to transform pixel colors in text mode, or even in modes 4 and 6, the CGA-compatible 4- and 2-color graphics modes. In order to do that you need to understand the format in which pixels come out of the VGA and into the DAC in those modes; once again, setting the palette to a pass-through state makes life easier.

In fact, you can also use color paging in any mode other than 256-color mode. The basic principle is that the color-processing steps that occur farther down

PRINTER GRAPHICS
IN
HIGH RESOLUTION!

GRAPHLINK™ Printer Graphics Toolkit

GraphLink™ lets your programs tap the *full potential* of printers to produce dazzling *printed* graphics in *high-resolution!*

- Uses the PRINTER'S resolution, not the SCREEN'S!
- *Supports the most popular laser, inkjet, and dot matrix printers*
- 80+ routines in a linkable library
- Fully scalable fonts
- *Emulates the Borland Graphics Interface - great for BGI programs!*
- Works with all screen graphics libraries
- Can *automatically* build the printer image
- *Supports Turbo Pascal 4.0 and higher*
- Available soon for Microsoft C, Quick C, and Turbo C
- *NO ROYALTIES!*
- Only ***$199*** (shipping: USA $5, Canada $10, Other $15) (with Source Code only $299)

30 Day No Risk Money-Back Guarantee

GRAPHLINK
The Printer Graphics Solution™

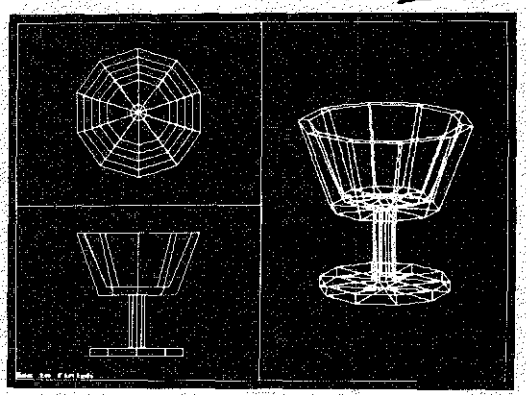
VISITECH SOFTWARE
D5 3807 Ridgewood Court
Pittsburgh, PA 15239 USA
(412) 733-4775

Circle **304** on Reader Service Card

THE ULTIMATE CAD/CAM ENGINE

TurboGeometry Library 3.0!

Helps You build it right THE FIRST TIME!



Over 300 2D & 3D device independent geometric routines. Surfacing; Spherical & Solid Geometry; Tangents; Curves; Hidden Line; Polygons; Perspectives; 2D & 3D Clipping; Link List Library & more. In C or Pascal, w/source. 400 pg manual. $199.95, Foreign $225.00. Prices incl S&H. 30 day guarantee. VISA, MC, PO, Chk.

2116 E. Arapaho Rd.
Suite 487
Richardson, TX 75081
USA

Disk Software, Inc.

FAX or Phone
214-423-7288
Toll Free
1-800-635-7760

Circle **229** on Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Develop Powerful Image-based Applications Quickly and Easily

with

VICTOR Image Processing Library

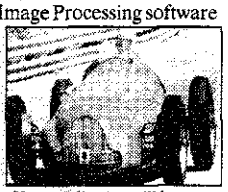
Powerful Image Processing

VICTOR simplifies development of scientific imaging, quality control, security, and image database software. Victor gives you powerful image processing (with functions like sharpen, outline, linearize, matrix convolution, overlay resize, flip, invert, mirror), screen display, and file handling capabilities.

Image-based applications can be developed in QuickC and Turbo C environments.

Gray Scale and Color Images

Victor allows you to handle 8-bit color and gray scale images from any source as TIFF/PCX/GIF files and display single or multiple images on EGA/VGA up to 800x600x256. Your software can have features such as: live video on VGA (frame grabber required), zoom, process and display multiple images, pixel editing and image processing. And, to get you up and running, we include Zip Image Processing software for rapid testing and prototyping of image processing and display functions. All processing functions are hardware independent. Graphics adapters supported but not required.



Your application will have powerful image handling capabilities.

Victor supports Microsoft C, QuickC, and TurboC, includes demonstration and prototyping software, and full documentation. Source code available. Victor Library $195.

Call (314) 962-7833 to order

VISA/MC/COD

Catenary Systems 470 Belleview St Louis MO 63119 (314) 9627833

Circle 321 on Reader Service Card

MORE 34010 Video Co-Processors from PC Tech

- Mono 736 x 1024 - 2048 x 1536
- Color 1024 x 768 (1280 x 1024 soon!)
- 512K - 4M DRAM
- Optional CCITT & DGIS support
- **AGE XoftWare X Window Server certified board**
- C Compiler, Assembler, Linker, Debugger, DGIS Development Kit & TIGA Toolkit available
- **TIGA** included with all boards
- Custom designs available
- **PC Tech understands the 34010**

For further information, please contact:

PCtech

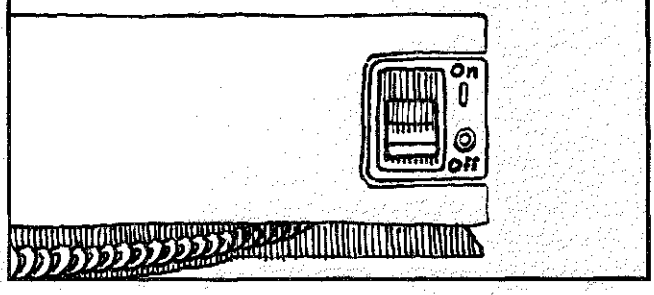
907 N. 6th St.
Lake City, MN 55041
(612) 345-4555 (Voice)
(612) 345-5514 (FAX)

PC Tech is a trademark of PC Tech, Inc.

PC, XT, AT, DGIS, XoftWare, and DGIS are trademarks or registered trademarks of their respective companies.

Circle 208 on Reader Service Card


```
   FP_SEG(ScreenPtr) = SCREEN_SEGMENT;
   FP_OFF(ScreenPtr) =(Y * SCREEN_WIDTH_IN_BYTES) + (X / 8);
#endif

   /* Set the bit mask within the byte for the pixel */
   outp(GC_INDEX + 1, 0x80 >> (X & 0x07));

   /* Draw the pixel. ORed to force read/write to load latches.
      Data written doesn't matter, because set/reset is enabled
      for all planes. Note: don't OR with 0; MSC optimizes that
      statement to no operation */
   *ScreenPtr |= 0xFE;
}

/* Draws a circle of radius Radius in color Color centered at
 * screen coordinate (X,Y) */
void DrawCircle(int X, int Y, int Radius, int Color) {
   int MajorAxis, MinorAxis;
   unsigned long RadiusSqMinusMajorAxisSq;
   unsigned long MinorAxisSquaredThreshold;

   /* Set drawing color via set/reset */
   outpw(GC_INDEX, (0x0F << 8) | SET_RESET_ENABLE_INDEX);
                                /* enable set/reset for all planes */
   outpw(GC_INDEX, (Color << 8) | SET_RESET_INDEX);
                                /* set set/reset (drawing) color */
   outp(GC_INDEX, BIT_MASK_INDEX);
                                /* leave the GC Index reg pointing to
                                   the Bit Mask reg */

   /* Set up to draw the circle by setting the initial point to one
      end of the 1/8th of a circle arc we'll draw */
   MajorAxis = 0;
   MinorAxis = Radius;
   /* Set initial Radius**2 - MajorAxis**2 (MajorAxis is initially 0) */
   RadiusSqMinusMajorAxisSq = Radius * Radius;
   /* Set threshold for minor axis movement at (MinorAxis - 0.5)**2 */
   MinorAxisSquaredThreshold = MinorAxis * MinorAxis - MinorAxis;

   /* Draw all points along an arc of 1/8th of the circle, drawing
      all 8 symmetries at the same time */
   do {
      /* Draw all 8 symmetries of current point */
      DrawDot(X+MajorAxis, Y-MinorAxis);
      DrawDot(X-MajorAxis, Y-MinorAxis);
      DrawDot(X+MajorAxis, Y+MinorAxis);
      DrawDot(X-MajorAxis, Y+MinorAxis);
      DrawDot(X+MinorAxis, Y-MajorAxis);
      DrawDot(X-MinorAxis, Y-MajorAxis);
      DrawDot(X+MinorAxis, Y+MajorAxis);
      DrawDot(X-MinorAxis, Y+MajorAxis);
      /* Advance (Radius**2 - MajorAxis**2); if it equals or passes
         the MinorAxis**2 threshold, advance one pixel along the minor
         axis and set the next MinorAxis**2 threshold */
      if ( (RadiusSqMinusMajorAxisSq -=
         MajorAxis + MajorAxis + 1) <= MinorAxisSquaredThreshold ) {
         MinorAxis--;
         MinorAxisSquaredThreshold -= MinorAxis + MinorAxis;
      }
      MajorAxis++;          /* advance one pixel along the major axis */
   } while ( MajorAxis <= MinorAxis );

   /* Reset the Bit Mask register to normal */
   outp(GC_INDEX + 1, 0xFF);

   /* Turn off set/reset enable */
   outpw(GC_INDEX, (0x00 << 8) | SET_RESET_ENABLE_INDEX);
}
```



PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

the pipeline toward the display can modify anything that comes earlier. As the last stage in the pipeline, the DAC can modify pixels in any mode at all. As the lead-in to the DAC stage, color paging can modify pixels in all modes except 256-color mode.

HOW LISTING 1 WORKS

Back to Listing 1. Once the circles are drawn, Listing 1 loads the upper 240 DAC locations with 240 colors, which are logically organized as 15 pages of 16 colors each, and then sets the palette RAM to a pass-through state. The palette RAM and the DAC must be set up **after** the mode set, because each mode set reprograms all of the palette RAM and at least a part of the DAC. During EGA-compatible mode sets, the first 64 DAC locations are programmed to match the EGA's set of colors; during mode sets for 256-color mode, all 256 DAC locations are programmed.

When everything is set up, Listing 1 enables 16-pages-of-16-colors paging, switches from the default color page, page 0, to page 15, and then starts flipping through the color pages (at the rate of 60 times per second) in the order 14, 13, and so on down to 1, and then back around to 15.

What does this accomplish? Well, the color pages were carefully selected as one-position rotations of increasingly bright green colors. That is, color page 1 translates a pixel value of 1 into dim green, a pixel value of 2 into brighter green, and so on up to a pixel value of 15, which is a very bright green indeed. Color page 2 is a one-position rotation of color page 1; in this color page, a pixel value of 1 produces next-to-dimmest green, a pixel value of 2 produces slightly brighter green, a pixel value of 14 produces brightest green, and a pixel value of 15 produces dimmest green. Each successive color page is a one-position rotation of the preceding page, except for color page 0, which isn't used because we need only 15 color pages to represent all possible rotations of a set of 15 colors.

CYCLING THROUGH

Cycling through these color pages causes the circles on the screen to appear to pulse outward. The net effect is one of smooth motion without a single pixel in display memory being modified. This is the wonder of color manipulation: effects worthy of high-end graphics systems with little effort. While there are limits to what color manipulation can do, what it does, it does very well.

Listing 1 can perform color paging using either the direct or the BIOS approach, depending on the setting of USE_BIOS. On my computer, both versions look exactly the same. Experiment for yourself and choose whichever you prefer, and please let me know about any quirks you encounter.

MORE COLOR PAGING

One advantage of color paging is that you can change the contents of non-displayed color pages without affecting the display at all. Manipulating the color pages in this way is much like page-flipping animation; while displaying one color page you alter another one, then display the altered page when it's ready to go. The benefit of this over simply reloading the DAC is that the user will see only the finished color configuration, not an intermediate state. This approach effectively expands the number of available color pages from 16 to more than 4 million (every combination of 16 colors possible on the VGA).

Listing 1 also illustrates the technique of changing a non-displayed color page so as to leave the display unaffected while the DAC is being loaded. At the outset of Listing 1, page 0 is displayed while the DAC locations for pages 1 through 15 are loaded. Once the loading is complete, then the program switches away from page 0 to begin cycling through the newly-loaded pages 15 through 1.

IN CONCLUSION

That brings us to the end of color paging, but certainly not to the end of color on the VGA. We'll begin the next installment with a look at color cycling. (I have yet to figure out a reliable way to load the DAC without glitching on all VGAs; does anyone have a handle on this? BIOS loading of the DAC seems prone to bounce and flicker, and direct loading has its risks as well.) As always, I'm open to suggestions, and in fact I've gotten some good ones lately. Keep those cards and E-mail messages coming. See you next time. PJ

CODE RUNNER®

Create ultra compact and fast TSRs with full DOS access!

The most powerful tool available for creating feature packed, tiny TSRs in C or assembler. Works with C compilers from Microsoft, Borland, Watcom and Zortec.

CodeRunneR Features

- Auto recovery of Initialization Code and Data
- All functions hand crafted highly optimized assembly language routines
- Function Level Granular
- Create MultiTasking programs
- Scheduler supports > 10,000 events
- Assess TSRs via up to 256, dynamic hot-keys or interrupts
- BCD Floating Point Library
- Source for TSRs: Help, Programmers Calculator, Business Calculator, ASCII/Color, Date/Alarm & Trap/Debug

CodeRunneR $149.00

Call our BBS for FREE sample TSRs, benchmarks & support. See for yourself how powerful and tiny CodeRunneR TSRs can be.
BBS: (508) 875-8009

NEW

PDK#3

- Spawn (EXEC) large programs from a hosting TSR (<1.5 KBytes)
- Page foreground application or hosting TSR to LIM* or Disk
- Pop-up TSRs over graphics screens!
- Save & Restore COM ports, Mouse and Math Co-Processor states.
- Turn ANY existing program into a TSR (without source code -- even Lotus 1-2-3)! * Needs PDK#1

PDK#1

- COM: Ultra fast interrupt driven multichannel COM library with background file transfers
- LIM: Transparently move TSR code &/or data to LIM
- Flexible printer SPOOLer
- Program registration and protection functions

PDK#1&3 $99.00 each

NO ROYALTIES. MONEY BACK offer and source code available.
Order today by Phone, FAX or BBS.
MasterCard, VISA & AMEX accepted.

Microsystems Software, Inc.™
600 Worcester Rd. Framingham, MA 01701
(508) 626-8511 FAX: (508) 626-8515

CodeRunneR is a Registered Trademark of Microsystems Software. 1-2-3 is a Trademark of Lotus.

Circle 40 on Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Choosing your database programming environment

The database dilemma

By Brett Glass

In the world of database application development, anarchy reigns. In many other areas of software development, it's relatively easy to choose the tools you need for a particular job. However, database software vendors offer an amazing array of toolkits, engines, languages, and application programs. After examining different database programming environments, I must admit that the picture is even stranger than I first thought: there is so little compatibility among or portability between the offerings of various manufacturers that the phrase "database standard" is actually an oxymoron.

I also found that database tool vendors too often shroud the inner workings of their products in a degree of secrecy that amounts to paranoia. If you call a vendor and ask even a simple question about what's going on inside a database program or "engine" that doesn't come with source code, odds are that you'll be stonewalled even if the answer is relatively straightforward. This developer-hostile attitude seems to be due to a belief (usually held by managers, not programmers) that revealing even a common-sense algorithm will give the competition an edge.

Brett Glass is a free-lance writer, programmer, and consultant residing in Palo Alto, California. He can be reached at rogue@well.sf.ca.us on Internet, on CompuServe at 72267,3673, or as glass on Bix.

PLUSES AND MINUSES

Each programming environment has its own advantages and disadvantages, and most—alas—can "lock in" the unwary programmer by making the resulting application partially or completely incompatible with everything else. To make sure that your development effort is successful, you must know what's out there and choose carefully—anticipating, as you do so, the future uses of your application and the databases it creates.

Describing all the possible options in detail in only a few pages is just not possible. Therefore, in this article I'll present an overview of currently available database tools and development strategies, along with the pros and cons of each category.

TOOLS OF THE TRADE

The tools you'll use to create and maintain a database system fall into several categories: source and object code libraries; fourth-generation languages and unique database languages; scripting languages (usually interpretive) which are built into existing database packages; third-party compilers that work with these languages; "engines" that let you access a popular database application's files from your own programs; and database server programs (usually also accessed via an object code library or an "engine").

In some cases, you may wish to start with nothing more than your favorite language compiler and a thorough knowledge of the problem at hand. This is the way to go if you need to squeeze every last ounce of performance out of your system. By maintaining complete control, you'll be able to optimize the code for size or speed and tailor every aspect of the database to your needs. You will own all the code outright; you won't have to worry about licensing problems, and your users won't need to buy any other software to use your code. Finally, you'll be familiar with all of the code should it require upgrades, bug fixes, or other maintenance.

ROLLING YOUR OWN

The disadvantage of rolling your own, of course, is that you'll spend a great deal of time "reinventing the wheel" when plenty of perfectly good database code already exists. If your application requires a friendly user interface, help screens, error checking code, or other important features, every hour you spend on the low-level database code may come out of the time required to polish these other parts of the program. Also, because your file format is likely to be unique, exchanging data with a mainframe or with a commercial database package will require extra work.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

For these reasons, I recommend coding your own database only if:

- your application is too simple to warrant the power of a complete database package;
- you need special features not available elsewhere; or
- you need every last ounce of performance the hardware can deliver.

Otherwise, I'd investigate one of the other approaches listed below.

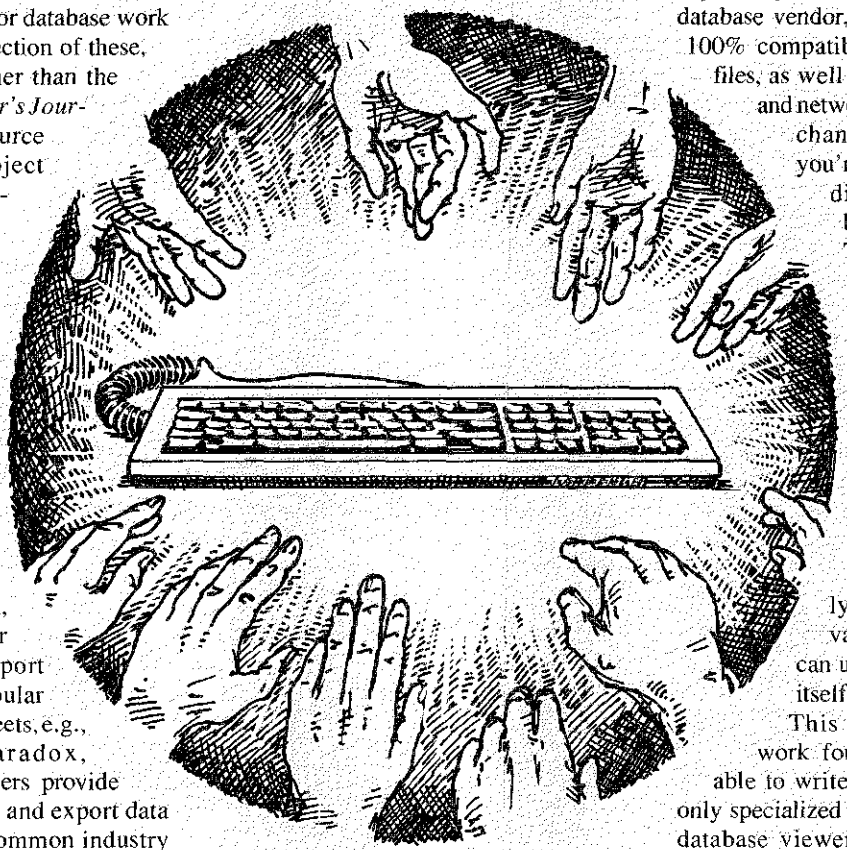
TOOLKITS AND LIBRARIES

Programmers' toolkits for database work abound. For a good selection of these, you need look no farther than the ad pages of *Programmer's Journal* to find dozens of source code toolkits and object code libraries that implement B-trees and B+ trees, as well as relational databases—all to be called from the language that you prefer. (The languages that are most commonly supported are C, C++, and Turbo Pascal, but if you do your homework you can also find them for BASIC, Ada, FORTH, Modula-2, and other languages. Some support the file formats of popular databases and spreadsheets, e.g., dBASE, Reflex, Paradox, Lotus, and Excel. Others provide code which will import and export data in these formats. It's common industry practice for these libraries to be provided for a one-time fee. Avoid any toolkit whose vendor does not explicitly state that you won't need to pay royalties on your finished product.

Because programmers' toolkits (hopefully) contain time-tested code, you can concentrate your testing efforts on other parts of the program. Examine the product carefully before incorporating it into your program. *Caveat emptor*—if your users discover bugs in the package, the errors will reflect on you, not on the library vendor.

When rating a library, look at the coding style if the source is available. Is it easy to read in case you need to change it? Is it easy to add extensions? How much memory does it consume? Does the publisher offer free technical support? Call the vendor and ask about updates and bug fixes. Is the code being maintained on a regular basis, or has the programmer left for an extended Hawaiian vacation? The more thorough your investigation, the less likely you are to be left in the lurch by a fly-by-night vendor.

I recommend using source or object libraries when:



- you have better things to do than drag out your copy of Knuth and write yet another B-tree implementation;
- you don't want to ask your users to buy a commercial application before they can run yours; and
- you don't need the special capabilities of an "engine," as detailed below.

DATABASE ENGINES

A database "engine" is like a toolbox in that it consists of an object module (usually without source code) that you link directly into your program. It's different, however, in that it's designed to work with the file format of an existing application, such as Borland's Paradox. It's usually supplied by the vendor of the database application in order to encourage customers to standardize on that application. (In the specific case of Paradox, a third-party vendor—TSR Systems Ltd., of Port Jefferson, NY—has announced that it will be releasing a competing engine as well as a compiler for Borland's Paradox Application Language.)

The advantage of using an engine, especially when it's supplied by the database vendor, is that it's likely to be 100% compatible with the database's files, as well as with its file-locking and networking facilities. (In fact, chances are that the code you're using was extracted directly from the database product itself.) The code is likely to be both more refined and more thoroughly tested than the best third-party library. And, so long as the database product is being maintained and upgraded, you are likely to be able to get support and bug fixes from the vendor. Finally—and this is a key advantage—your customers can use the database product itself to query your files. This may mean a lot less work for you, since you'll be able to write a program to perform only specialized tasks. The application's database viewer and report generator can handle the more mundane stuff, and their manuals, tutorials, and technical support staff will be available to assist the users when they get into trouble.

ENGINE TROUBLE

The downside of engines is that the vendor has no vested interest in allowing you to port your code to or connect with other database products. Each engine has a completely different API, so you may have to do some serious rewriting if you change database products. And if you want to be able to access different vendors' databases at the same time, you'll have to use multiple toolkits or engines in your product.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

A scripting language is of little use if it doesn't have sufficient power for the task at hand.

An engine may also provide only a subset of the database product's capabilities. For example, while Borland's Paradox engine is an excellent way to access Paradox files, it doesn't allow you (as Paradox itself does) to write embedded SQL queries and pass them to a database on a server. The final important drawback of engines is that you don't have and won't get source code. If you want to add a capability that isn't there, such as putting pictures or text strings longer than 256 bytes into a Paradox database, you may be out of luck—or you may have to do something awkward to squeeze the data in.

Still, I favor database engines for many of the applications I write. I'd recommend that you use them when:

- the main purpose of your code is to automatically add data to, or extract data from, the database; and
- it's acceptable to tell users to invoke the packaged version of the database program when they wish to manually query, view, or generate reports from the database.

SCRIPTING LANGUAGES

If you expect all of your users to have copies of a particular database package, and you know—or are willing to learn—that package's scripting language, you may want to use that language to build your program. Some scripting languages, such as dBASE and PAL, have become so popular that there are dozens of books devoted to programming tricks and tips.

The most important advantage of these languages is that they're tailored to the job at hand. When using a toolkit or an engine, you constantly deal with making a very general-purpose programming language do the job of a special-purpose one, and the process can be awkward. In fact, the APIs of engines and toolkits are usually much more complex than the commands in any special-purpose database language. An additional advantage is that database vendors have strong incentives to maintain compatibility with your code in future releases of their product. By contrast, an engine or third-party toolkit might be "broken" if the vendor introduces a new file format or adds new options. Finally, you may be able to take advantage of high-level tools such as screen generators that automatically generate scripts for database programs.

A scripting language is of little use if it doesn't have sufficient power for the task at hand. If you need to perform numerical operations which are best done in C, for example, but the scripting language doesn't provide an "escape hatch" that lets you execute code written in this language to get the job done, you may be better off with a toolkit or an engine. A scripting language may also be significantly slower than a compiled language for many applications, once again making a toolkit or engine the better choice.

Two more potential disadvantages of using a scripting language are that, in many cases, portability is close to nil, and that your customers will need the database package to run your code. I call these "potential" disadvantages for a reason, however: they sometimes disappear when a database package becomes popular. Several compilers are available to create stand-alone, royalty-free applications from dBASE scripts, and some database packages are able to read scripts from others.

Nonetheless, because of the specialized and proprietary nature of scripting languages, I would recommend that you use them only if:

- you're willing to make a long-term commitment to a database package or its work-alikes; and
- the language is robust enough for your programming needs.

4TH-GENERATION LANGUAGES

If you don't mind learning a whole new language, there's a lot to be said for another approach: generating your application using a "4GL" (4th-Generation Language). Clarion Professional Developer, for example, provides you with a complete environment in which to develop stand-alone applications, including a screen generator and extensions to handle things like serial communications. Others compile into the scripting languages of popular database packages, as noted above.

In general, if the 4GL you choose is designed to handle the kind of problem you're trying to solve, you should be able to get the job done in record time—probably faster than with any other set of tools mentioned here. The main pitfall to watch out for when using 4GLs, however, is a runtime fee. If you have to pay a royalty on every copy of your application or sell a runtime package with it, it may hurt your sales. Another drawback of 4GLs is that their file formats are often unique; you can't read them with any other product.

If you're interested in using a 4GL, get as much information as possible before buying—including demo disks, live demonstrations, and sample code. If one of the sample applications looks like what you're trying to produce, and you don't need to be compatible with anything else, go for it! Many of these environments make it a joy to program.

PICK AN OPTION

A few users would give me a very hard time if I didn't mention the Pick Operating System as an option for developing database applications. This unique environment relies on Pick BASIC, a BASIC with database extensions, to get the job done. The operating system itself is multi-user and multitasking and can run under UNIX.

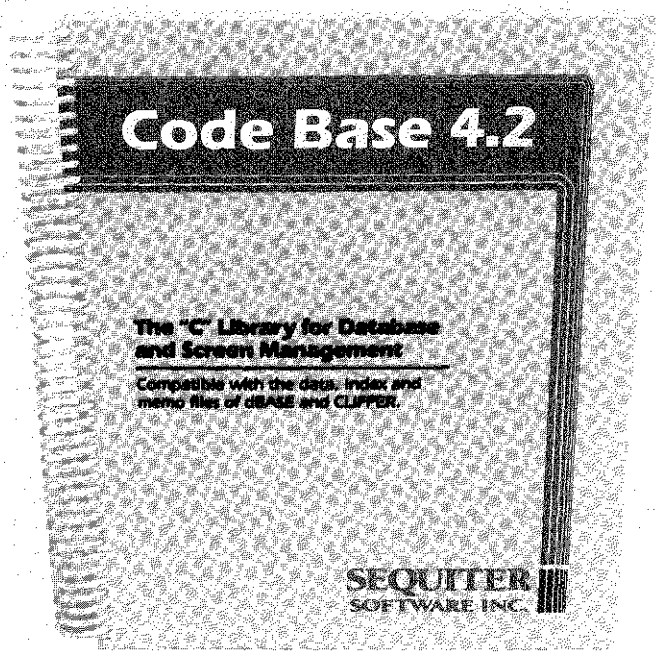
Whether or not you want to use a complete operating environment plus a unique, non-portable language is somewhat of a religious issue. Fanatical Pick users swear by it; others swear at it. I haven't used Pick myself, but it's a candidate worth investigating if you're working on a system that's already running UNIX, or if you have access to a local Pick User's Group.

CLIENT/SERVER SYSTEMS

Client/server databases come in two pieces: a high-powered back end, which

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

C Speed
C Portability
C Flexibility


Code Base 4.2
The "C" Library for Database and Screen Management
Compatible with the data, index and memo files of dBASE and CLIPPER.
SEQUITER SOFTWARE INC.

dBASE Power

Build a multi-user, dBASE compatible application which is several times faster than dBASE IV, Clipper or Fox Pro. Watch its windows and menus appear instantly on any computer.

Portable

Port your application to any environment with a C or C++ compiler. Access megabytes of memory using 386 DOS compilers, OS/2, Unix or Microsoft Windows.

Compatible

As you directly use the data, index and memo files of dBASE III through IV or Clipper, you can use Code Base 4.2 with any dBASE compatible product.

Easy

Consult examples in the 280 page user's guide as you interactively execute Code Base 4.2 routines from a learning utility. You will remember the routines which are named like dBASE commands.

Small

Make stand alone executable files as small as 14K. Code Base 4.2 executables are ½ to ⅓ the size of corresponding Clipper executables.

Complete

Enjoy the benefits of complete dBASE functionality, including browse, edit, menus, windows, multiple index files per database, dBASE expression evaluation, relations and filters.

Order Today

Order the DOS-OS/2 version for $295. Call (403) 448-0313 or fax (403) 448-0315. Discover why Sequiter Software Inc. and most software dealers offer a 60 day money back guarantee. Source is included and there are no royalties!

Sequiter Software Inc. • P.O. Box 5659, Station L, Edmonton, Alberta, Canada T6C 4G1

Circle 269 on Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Easy
CASE
$295.00 * Plus™
New Version 2

- Structured Analysis
- Structured Design
- Data Modeling
- Real Time Support
- Data Dictionary

Diagrams Supported:
- Data Flow Diagrams
- Structure Charts
- State Transition Diagrams
- Data Model Diagrams
- Entity Relationship Diagrams

Methods Supported:
- Yourdon/DeMarco
- Gane & Sarson
- Ward-Mellor/Hatley
- Yourdon/Constantine
- Chen, Martin, Bachman

Features:
- Export/Import DD to dBase III Plus & SDF
- Chart Linking
- Integrated Process Specification Editor
- Extensive Printer, Plotter and Desktop Publishing Support
- Integrated Data Dictionary, Manager and Reports
- And More . . .

Runs On:
IBM PC, Hercules, EGA, VGA, Mouse, 640 K RAM.

Professional Pack $395.00
(Includes DFD Level Balancing and Diagram Analysis Module)

Evergreen CASE Tools

16710 N.E. 79TH Street
Suite 104
Redmond, WA 98052
(206) 881-5149

* Plus Shipping & Handling: USA $5, Canada & Mexico $10, other Countries $25. dBase III Plus is trademark of Ashton-Tate.

There's a real danger if the system that created the lock file goes down or loses network access

fields and responds to queries, and a front end, which may be a packaged program or an application that you write. The front ends access the back via a network or via the interprocess communications facilities of a multitasking operating system. Among the most heavily hyped client/server systems are the Microsoft/Ashton-Tate/Sybase SQL Server, which runs under OS/2, and Oracle, which runs on many platforms, including VMS and UNIX.

If you're writing your client application in a standard programming language, you'll usually access a database server via an API provided for that purpose by the vendor of the server software. As with other APIs, the calls will probably not be portable. However, if they contain embedded SQL commands that stay close to the ANSI SQL standard, then the text of the commands may be usable on other platforms.

Client/server systems aren't the only kind of networked database. Other products, such as Paradox, implement networking by using a shared file system with file and record locking. However, because most file sharing facilities aren't specifically tuned for use with database queries, this method tends to be slower and somewhat more cumbersome. What's more, if a networked database uses the existence of a file (often called a "lock file") as a semaphore to restrict access to one user, there's a real danger if the system that created the lock file goes down or loses network access: the database may remain locked (and therefore unusable) until an administrator intervenes.

If you know that your database is going to require network capabilities or be used by large numbers of people at once, you'll want to investigate client/server systems. I'd recommend them for any system which is large, networked, or transaction-based, and for any application in which SQL is required.

With so many options, it's no wonder that programmers have a difficult time choosing among the many ways of implementing database software. No matter which solution you choose, you risk getting stuck with your own incompatible system, or being tied to a single vendor's software or hardware.

"UNIVERSAL" TOOLKITS

The answer to this problem may come in the form of "universal" database toolkits—modules which allow you to access many client/server systems and proprietary file-based systems via the same API. Under such a system, you could add modules to handle new databases or updated versions of existing ones without changing any of your own code. Ideally, the modules could be loaded as TSRs independently from your application, so that it would not even be necessary to ship a new version of the program, just the new TSR.

I know of no products currently shipping that implement such a scheme, so this represents a hot opportunity for all the budding entrepreneurs out there who are reading this. The first vendor who can find a way to give users transparent access to dBASE and Paradox files, as well as to SQL Server, Oracle, and IBM mainframe databases is ahead of the game—the one who can do it all through the same uniform, standard API wins the prize!

GOOD LUCK, LANCELOT

As I mentioned at the beginning of this article, I do not envy any programmer who is about to dive into a new database programming project. Unlike most other realms of programming, databases are an area in which incompatibility is the norm and source code portability is as distant a goal as the Holy Grail. Don't despair, however: armed with a basic knowledge of what's available, you should be able to make a prudent choice that will serve you—and suit your customers—until there's more standardization in this confusing corner of the computer industry. PJ

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Finally, a fast, powerful text editor that integrates your favorite programming tools and uses no memory!

INSTALL.C
Command Mode
Help
VEDIT COMPILER SUPPORT Ver 1.40
Support Microsoft C 5
Options /AL /Zm
Debugger CodeView
Debugger Opt /43 /F
Make Options -d -p -fmakefile
Project File D:\INSTALL\INSTALL.C
Edit File D:\INSTALL\INSTALL.C
Default Ext Edit=*.C Proj=*.C
E. Edit current file S. Select compiler
N. Edit new file O. Change options
P. Specify project G. Goto DOS
C. Compile project R. Recognize errors
D. Run Debugger Q. Quit support
M. Run Make X. Exit VEDIT
Enter letter:
Select desired option

VEDIT PLUS

FREE Fully Functional Evaluation Copy Call 1-800-45-VEDIT

- **Mouse Support**
- **Pull-down menus**
- **Columnar blocks**
- **1000 Level Undo**
- **Regular expressions**
- **Small 70K size, fast**
- **Emulate Wordstar, Word Perfect, Brief, vi, others**
- **Edit text and binary files of any size and line length**
- **Powerful macro programming language for instant "off-the-cuff" macros**

DOS XENIX UNIX QNX FlexOS

The new VEDIT family of text editors offers stunning performance, versatility and ease of use. Completely written in assembly language, they are small and lightning fast. (3 to 30 times faster than other editors on large files where speed really counts.) Edit text and binary files of any size, even 100+ megabytes. Installation is trivial; VEDIT.EXE and an optional help file are all you need - no overlays, no configuration files, no environment variables.

For programmers, the new compiler support in VEDIT and VEDIT PLUS is a breakthrough. Run not only popular compilers, but debuggers and your favorite tools from within the editor. When shelling to DOS, VEDIT swaps itself and any desired TSRs out of memory to give you more memory than when you entered VEDIT. Only VEDIT offers you the advantages of a powerful editor without giving up the convenience od an integrated environment.

Call for your free, fully functional, evaluation copy today. See why VEDIT has been the choice of 100,000 programmers, writers and engineers since 1980.

VEDIT Jr. - Unmatched performance for only $29.

All VEDIT editors include a pull-down menu system with "hot keys", context sensitive help, pop-up status and ASCII table, a configurable keyboard layout and flexible, unlimited keystroke macros. Perform block operations by character, line, file or column. Undo up to 1000 keystrokes - keystroke by keystroke, line by line, or deletion by deletion. Automatic indent, block indent and parentheses matching speed program development. Word wrap, paragraph formatting, justification, centering, adjustable margins and printing for word processing. Run DOS programs.

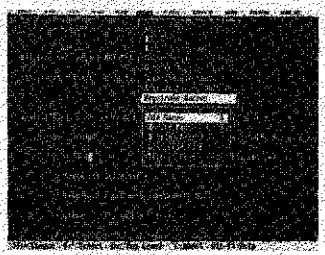
VEDIT - A best value at only $69.

VEDIT can simultaneously edit up to 36 files and split the screen into windows. Search/replace with regular expressions. The most integrated compiler support available. Run VEDIT PLUS macros.

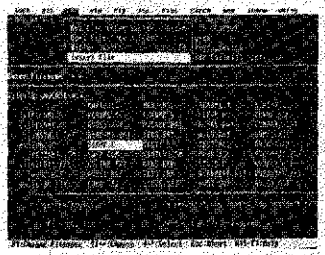
VEDIT PLUS - Ultimate programmer's tool for only $185.

VEDIT PLUS adds the most powerful macro programming language of any editor. It eliminates repetitive editing tasks and permits creating your own editing functions. The macro language includes testing, branching, looping, user prompts, keyboard input, string and numeric variables, complete control over windows plus access to hardware interrupts, memory and I/O ports. Source level macro debugging with breakpoints and tracing.

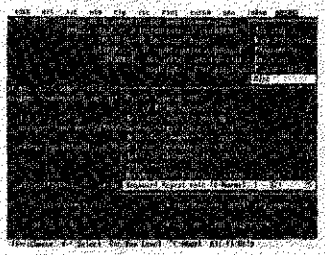
VEDIT PLUS - $185 for DOS, $285 for UNIX/XENIX. 30 Day Money-back guarantee. Discount pricing for multiple users, schools and OEMs. VEDIT Jr. site licenses start at only $250.



An intuitive user interface with pull down menus, hot keys, mouse support and context sensitive help make VEDIT easy to use, easy to learn.



Point and shoot file selection makes it a snap to edit new files, merge and split files, run macros and much more. Select files with the cursor keys or the mouse.



Just about everything about VEDIT is configurable, from the keyboard layout, to the screen colors, to the way control characters, tabs and the end of lines are displayed. Configure VEDIT with easy to use menus.

BENCHMARKS IN 3 MEG FILE

	VEDIT	BRIEF	Sage	QEdit
Simple search	57 sec	1:17 min	28 sec	Cannot
Save and continue	52 sec	3:52 min	1:47 min	Cannot
Load, modify, save, exit	21 sec	49 sec	1:38 min	Cannot
Block-column copy (40x200)	2 sec	30sec	2 sec	2 sec
Delete one column in file	9:58 min	**1:50 hour**	**1:03 hour**	Cannot
60,000 replacements	3:18 min	**1:44 hour**	**1:32 hour**	Cannot

VEDIT is a registered trademark of Greenview Data, Inc. BRIEF is a trademark of UnderWare, Inc. Sage Profession Editor is a trademark of Sage Software Inc. QEdit is a trademark of SemWare.

* DOS version also supports Concurrent DOS, DESQview, Microsoft Windows, PC-MOS/386 and most networks. Special CRT terminal version runs over serial lines and modems.

* Free evaluation disk is fully functional and can edit small files.

Greenview Data

P.O. Box 1586, Ann Arbor, MI 48106
(313) 996-1299, Fax (313) 996-1308

Circle 56 on Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Networking databases

Converting a single-user dBASE system to multi-user

By Lea C. Curtis

Networks are becoming increasingly common. In more and more cases, PC systems need to allow multiple users to access the same data from separate PC workstations. In some cases, a system originally designed for a single user must now allow several people to access and update data simultaneously. Rather than starting from scratch, it makes sense to convert an existing single-user system to a multi-user one.

Multi-user programming adds concepts such as *locks* and *exclusive use*. Once a user has exclusive use of a file or has locked a file, every other user who needs this file must wait until the original user is finished. Although there are situations that require exclusive access, it should normally be avoided so that other users are not forced to wait unnecessarily. Of course, deciding what is a valid wait is a judgment call on the part of the multi-user system designer.

dBASE, FoxBASE, Clipper, and compatible products all provide multi-user commands to support locking, unlocking, and exclusive use. We'll look at when to use and when to avoid each of these specialized commands.

Lea C. Curtis is a dBASE and COBOL programmer/analyst. Her work has appeared in Data Based Advisor *and* New Jersey Outdoors. *She can be reached at the A.M. Best Company, Ambest Road, Oldwick, NJ 08858.*

When converting an existing single-user system to a multi-user one, start by locating all occurrences of commands which require exclusive use or locks. A utility such as GREP is handy for this. After these occurrences have been located, analyze each to decide whether the command in question must be used or can be avoided and replaced with comparable logic.

EXCLUSIVE USE

Exclusive use prevents any other users from reading or writing to a file. Accordingly, avoid such a restrictive mode if at all possible. File locks, whether they are implicit or explicit, allow other users read but not write access to a file. This situation provides more flexibility but should still be used sparingly. Implicit and explicit record locks are usually not a problem, since it is unlikely that two users will require access to the same record at the same time on a regular basis; however, such locks should be held for the shortest possible amount of time.

Scan the entire system for occurrences of **set exclusive on** and **set exclusive off**. **Set exclusive on** will lock every database opened after it is issued until a **set exclusive off** command is issued. There should be one **set exclusive off** command in the top program of the system and no **set exclusive on** commands elsewhere in the system. Instead, use file locks on individual databases in order to avoid exclusive file use where not absolutely necessary.

Exclusive use, which is obtained by opening databases exclusively or **setting exclusive** use **on**, is required with the commands in Table 1. These commands do not permit file sharing, either for reading or for writing, because they need access to all records and to the file header. There are ways to make using these commands smoother in the multi-user environment, however, which we will discuss below.

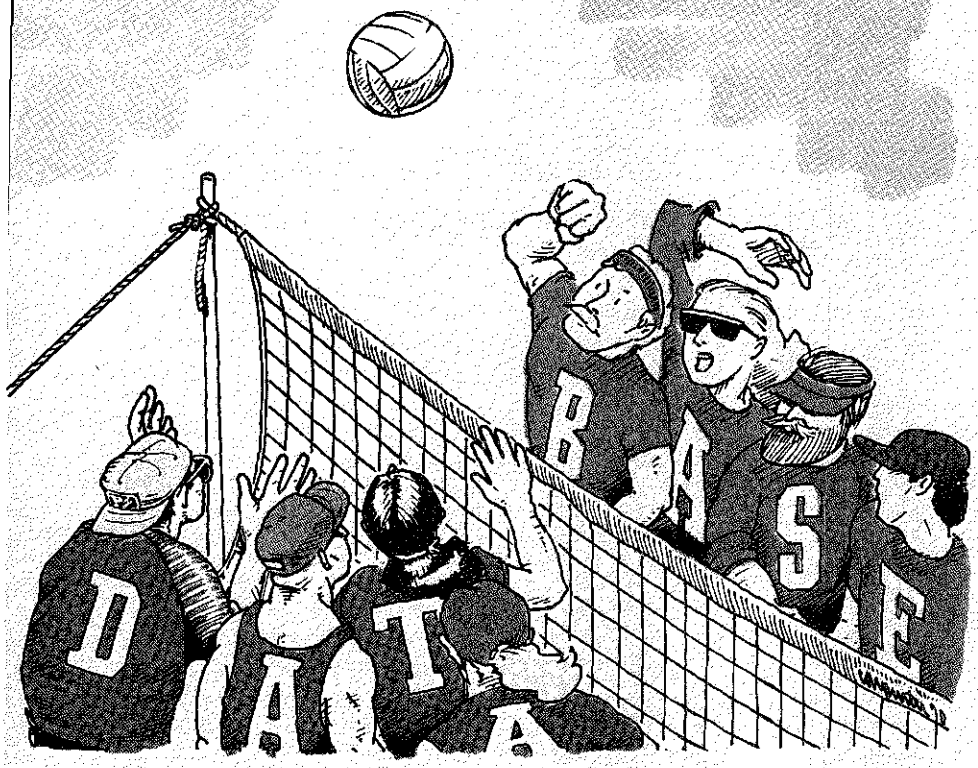
If **insert** is used on an indexed file, the record will be added to the end of the file and the header will be locked, making **insert** equivalent to an **append blank**. Ways to optimize **append blank** are discussed below.

Pack can be avoided by **setting deleted on** at the beginning of a program, which will bypass deleted records, and then **setting deleted off** when exiting the program. Deleted records will accumulate in a file over time, so perform

TABLE 1: Exclusive use

```
insert [blank]
modify structure
pack
reindex
zap
```


PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED



a **pack** occasionally to permanently remove the records marked for deletion by the **delete** command. This will free up the space used by these unwanted records and reduce the likelihood of seeing a DOS "out of space" message.

The frequency of a pack depends on how often records are deleted and how tight space is on the disk. To accommodate a pack of your files which could be run by the user without your intervention, have a "Pack Databases" option on a maintenance menu. Be sure to **set deleted off** before using a **recall** command or making a **deleted** test, then set delete **on** again afterwards.

REINDEXING

Reindex, which updates indexes, can be avoided by keeping indexes open whenever they could change. Any time that records will be added to or **pack**ed from the file, or when the file will be **zap**ped, all indexes that are related to the database should be open so that they can be updated. In addition, if a key field for an index could change via a **replace**, then the affected index should be open.

It is also wise to have a "Reindex Databases" option on a maintenance menu to allow recreation of all indexes using **index on...**. In this way, indexes can be easily corrected or even recreated—by the user—in case they are somehow deleted or damaged. Keep in mind, however, that there may be limits on the number of indexes and files that can be maintained by the system at a given time. dBASE III and FoxBASE, for example, allow only seven open indexes per database, and a total of 21 open indexes at the same time. More than that generates a "Too many files open" error.

Using **zap** to clear out databases can be avoided by keeping an empty copy of the database (for its structure) and using **copy** to write the empty database over the non-empty one. Remember that if **safety** is set **on**, dBASE and FoxBASE will ask the user whether or not to overwrite the file—a potential source of confusion. Avoid the problem by issuing a **set safety off** before copying the file, then **set safety on** afterward.

IMPLICIT FILE LOCKING

Implicit database locking occurs with the commands in Table 2. In these cases, there is no need to issue a file lock since it is automatic; however, a file **unlock** must be issued to free the file for other users. Remember that these file locks prevent other users from writing to, but not from reading the file. Many of these commands can be avoided by using a **do while .not. eof** loop and testing for different conditions. Be sure to go to the top of the file before entering the **do while** loop.

Append from can be eliminated by using an individual **replace next** *n* for each of the fields common to both databases from within a **do while .not. eof** loop. After the **replace next** *n* commands, an **unlock** command must be issued for each record, since **replace next** *n* locks the record. If the **append from** has a **for** clause, this can be simulated by testing for the condition via an **if...endif** construct within the **do while** loop.

A read-only **browse** (with **noappend** and **nomodify** options specified in FoxBASE) can be avoided by copying the database to a unique temporary file, browsing this file, and then deleting the temporary file. In order to create a unique filename, values such as time, date, and station number can be used. In FoxBASE, the **sys(3)** function creates a unique filename. If an index file is needed, use the unique file exclusively and create a temporary index (with the same unique name) with **index on...**. Browses which allow the user to modify the file are dangerous since the edits cannot be controlled. Avoid them if at all possible.

Copy to can be replaced in a way similar to **append from**. Start with a **copy structure** from the sending database to the receiving database, then use **replace next** *n* for each field within a **do while** loop.

Count to can be simulated with the **reccount** function. If there is a filter in use, **count to** can be simulated by going to the top of the file, then incrementing

TABLE 2: Implicit file locks

```
append
append from
average *
browse
catalog *
copy [structure] *
count *
delete next n/all for ... while
   ... (with scope)
index on...
join
recall next n/all for ... while
   ... (with scope)
replace next n/all for ... while
   ... (with scope)
sort *
sum *
total *
update

[* These commands do NOT implicitly
lock databases in FoxBASE.]
```


PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

A situation where two people could change the same field at the exact same time is not desirable.

a counter while skipping through the file, all within a **do while** loop.

Delete, **recall**, and **replace** with the **next** *n*/**all** or **for...while** parameters can be converted to individual record commands within a **do while .not. eof** loop. Note that with the above parameters, **delete**, **recall**, and **replace** lock records (as is required in a multi-user system), and thus require a subsequent **unlock**. Instead of using a **for** clause, the condition can be checked with an **if...endif** construct within the **do while** loop.

Another method to bypass a **for** clause is to issue a **locate for** command before the **do while...enddo** loop and then use a **continue** command within the loop. The problem with **locate**, however, is that you pay a penalty in time when running over a network or on large database files. Depending on the network configuration and/or the size of the database, the delay may be unnoticeable or excessive.

INDEXING

Index on... can be avoided by making sure that all indexes have been created, either before delivery of the system or by means of a **reindex** program (see **reindex** above). As mentioned before, indexes can change as a result of adding records to, **pack**ing, or **zap**ping a database. Take care to open all indexes related to the database before using any of these commands.

In addition, if a key field is **replaced**, the value of this key could change, so the affected index should be kept open. As noted above, have a facility to re-create all indexes should they be deleted or damaged.

TABLE 3: Implicit record locks

```
append blank **
delete [record n/next 1]
gather
recall [record n/next 1]
replace [record n/next 1]

[** Append blank implicitly locks
the record being added and also
locks the database header, i.e.,
two users cannot append blank at
the same time.]
```

IMPLICIT RECORD LOCKING

The commands listed in Table 3 implicitly lock records. When **append**ing a new record and **get**ting database field values, it is better to **get** the field values to memory variables, **read**, **append blank**, and then **replace** the database fields with the memory variables. This way, the record is not locked while waiting for user input. **Append blank**, **delete**, **recall**, and **replace** are all quick commands, so record locks from these are brief enough to be innocuous.

EXPLICIT LOCKING

Avoid explicit locking as much as possible, since other users will be forced to wait. Issue explicit file or record locks via **flock**, **lock**, or **rlock** when exclusive access is necessary—when the data has to be accurate and timely.

Replace and **@...get...read** of database fields require explicit record locking and subsequent unlocking. A situation where two people could change the same field at the exact same time is not desirable. If a database has been opened exclusively, explicit locking is not necessary since the entire file is already locked. For all other cases, an error routine is needed so that record locks will be automatically requested when error 130 occurs (see below). Explicit **unlock**s must be coded directly after issuing **replace**s or **@...get...read**s of database fields in the program which issued them. An **on error** routine should not do the unlocking.

When using **@...get...read** to obtain database field values, it is better to **get** the values to memory variables, as mentioned previously. This keeps the record unlocked while waiting for user input.

ON ERROR ROUTINES

Four error numbers, returned from the dBASE **error** function, can occur in a multi-user environment:

- 108 - attempting to lock a file that is already locked by another user;
- 109 - attempting to lock a record that is already locked by another user;
- 110 - exclusive use of file is required for command;
- 130 - record is not or cannot be locked.

An **on error** routine (see Listing 1) can use the error number to decide what kind of lock is being requested and then can automatically try to satisfy the lock request.

The **on error** routine continually attempts to obtain the required lock by issuing the appropriate locking function within a **do while** loop. Use a counter to prevent an endless wait in the loop. If the desired lock has not been obtained after a specified number of attempts, it's best to ask the user whether to keep trying for the lock or to terminate the command the program just issued.

During each attempt, the sample **on error** routine issues a **retry** command which returns to the calling program and re-executes the command which resulted in an error. If the lock can be obtained and no error results, the calling program proceeds normally. If the lock cannot be obtained (after a specified number of retries) and the user decides to stop trying, the **on error** routine will close databases, terminate the calling program, and return to the master program—the top program in the system. Here, the user decides whether waiting is worse than losing the update information just issued. Obviously, it is very important to make the user wait as infrequently as possible.

Note that the **retry** command differs from a **return** in that the instruction that caused an error and subsequently called the **on error** routine is the one executed upon return, not the instruction which follows it. Error 110—exclusive file use is required—is handled differently than the other errors. The user is informed that file use must be exclusive for the given command; then the databases are closed, the calling program is terminated, and control is returned to the master program.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

AT TSR SYSTEMS
WE SPEAK YOUR LANGUAGE!

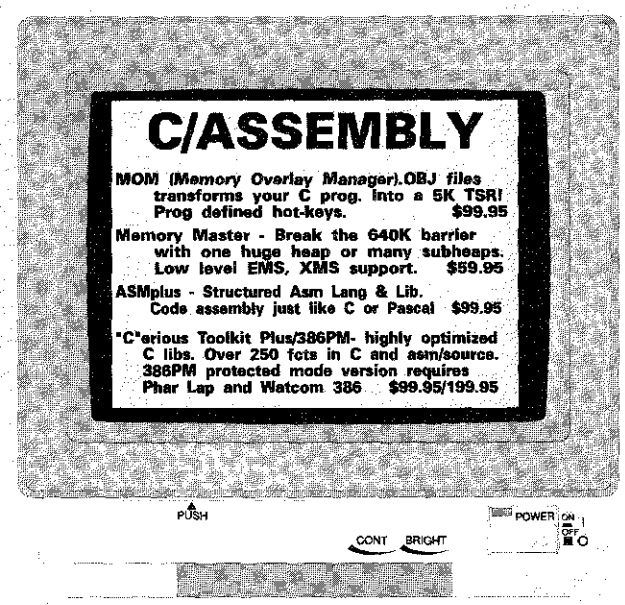
C/ASSEMBLY

MOM (*Memory Overlay Manager*).OBJ files transforms your C prog. into a 5K TSR! Prog defined hot-keys. $99.95

Memory Master - Break the 640K barrier with one huge heap or many subheaps. Low level EMS, XMS support. $59.95

ASMplus - Structured Asm Lang & Lib. Code assembly just like C or Pascal $99.95

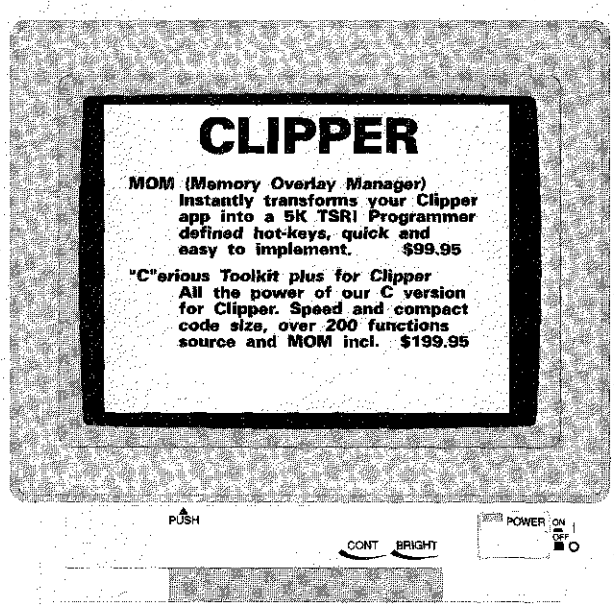
"C"erious Toolkit Plus/386PM- highly optimized C libs. Over 250 fcts in C and asm/source. 386PM protected mode version requires Phar Lap and Watcom 386 $99.95/199.95



CLIPPER

MOM (*Memory Overlay Manager*) Instantly transforms your Clipper app into a 5K TSR! Programmer defined hot-keys, quick and easy to implement. $99.95

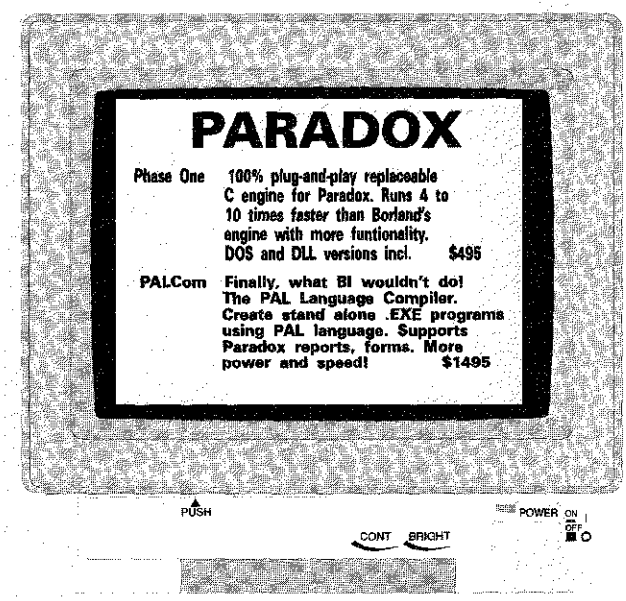
"C"erious Toolkit plus for Clipper All the power of our C version for Clipper. Speed and compact code size, over 200 functions source and MOM incl. $199.95



PARADOX

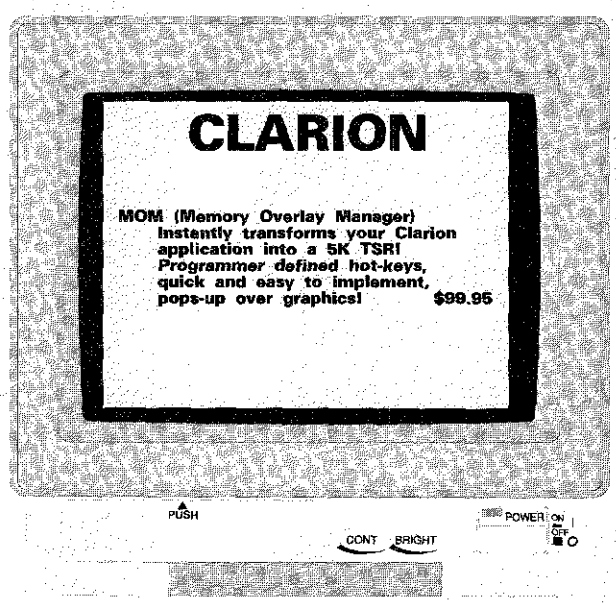
Phase One 100% plug-and-play replaceable C engine for Paradox. Runs 4 to 10 times faster than Borland's engine with more funtionality. DOS and DLL versions incl. $495

PALCom Finally, what BI wouldn't do! The PAL Language Compiler. Create stand alone .EXE programs using PAL language. Supports Paradox reports, forms. More power and speed! $1495

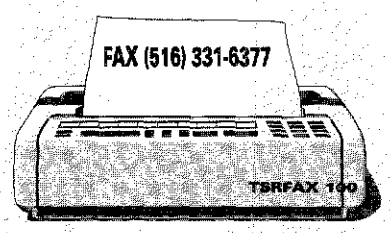


CLARION

MOM (*Memory Overlay Manager*) Instantly transforms your Clarion application into a 5K TSR! *Programmer defined hot-keys*, quick and easy to implement, pops-up over graphics! $99.95



TSR SYSTEMS LTD.
116 Oakland Ave.
Port Jefferson, NY 11777
Phone (516) 331-6336



BBS
(516) 331-6682

FREE DEMO PROGRAMS

Circle 72 on Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

LISTING 1

```
* Errorprg.prg - Multiuser Error Recovery Program
parameter err_no,timeout,row1,col1,row2,col2

* Please make row2 at least 2 greater than row1 and col2 at least
* 60 greater than col1 for the window

do case

case err_no=108          && File locked by another user
   * This occurs when these commands are issued:
   * APPEND, APPEND FROM, BROWSE, DELETE, INDEX ON
   * JOIN, RECALL, REPLACE, UPDATE

   if .not. flock()
   * display window - trying to access file
     save screen to screl
     @row1,col1 clear to row2,col2
     @row1,col1 to row2,col2 double
     @row1+1,col1+10 say "Attempting to access file...please stand by"

     timer=1
     do while .not. flock() .and. timer <= timeout
       timer=timer+1
     enddo
     if timer < timeout
       restore screen from screl
       retry
     else
       @row1+1,col1+10 say space(50)
       @row1+1,col1+10 say "Unable to access this file now..."+;
          "Retry File Lock (Y/N)?"
       clear typeahead
       key = 0
       do while key = 0
         key = inkey()
       enddo
       keyval = upper(chr(key))
       restore screen from screl
       if keyval = "N"
         clear gets
         clear typeahead
         close databases
         return to master
       else
         retry
       endif
     endif

   else
     retry
   endif


case err_no=109          && record locked by another user
   * This occurs when these commands are issued:
   * APPEND BLANK, DELETE, GATHER
   * RECALL, REPLACE

   if .not. rlock()
     * display window - trying to access record
     save screen to screl
     @row1,col1 clear to row2,col2
     @row1,col1 to row2,col2 double
     @row1+1,col1+10 say "Attempting to access record...please stand by"
     timer=1
     do while .not. rlock() .and. timer <= timeout
       timer=timer+1
     enddo

     if timer < timeout
       restore screen from screl
       retry
     else
       @row1+1,col1+10 say space(50)
       @row1+1,col1+10 say "Unable to access this record now...+;
          "Retry Record Lock (Y/N)? "
       clear typeahead
       key = 0
       do while key = 0
         key = inkey()
       enddo
       keyval = upper(chr(key))
       restore screen from screl
       if keyval = "N"
         clear gets
         clear typeahead
         close databases
         return to master
       else
         retry
       endif
     endif

   else
     retry
   endif


case err_no=110          && exclusive file use required
   * This occurs when these commands are issued:
   * INSERT [BLANK], MODIFY STRUCTURE, PACK
   * REINDEX, ZAP

   * display user window - trying to access file
   @row1,col1 clear to row2,col2
   @row1,col1 to row2,col2 double
   @row1+1,col1+10 SAY "EXCLUSIVE use of file required. Press any key to
exit."
   clear typeahead
   key = 0
   do while key = 0
     key = inkey()
   enddo
   keyval = upper(chr(key))
   clear gets
   clear typeahead
   close databases
   return to master


case err_no=130          && record is not locked.
   * This occurs when these commands are issued:
   * REPLACE
   * READ (database fields)

  if .not. rlock()
  * display window - trying to access record
    save screen to screl
    @row1,col1 clear to row2,col2
    @row1,col1 to row2,col2 double
    @row1+1,col1+10 SAY "Attempting to access record...please stand by"
    timer=1
    do while .not. rlock() .and. timer <= timeout
      timer=timer+1
    enddo

    if timer < timeout
      restore screen from screl
      retry
    else
      @row1+1,col1+10 say space(50)
      @row1+1,col1+10 say "Unable to access this record now...+;
          "Retry Record Lock (Y/N)? "
      clear typeahead
      key = 0
      do while key = 0
        key = inkey()
      enddo
      keyval = upper(chr(key))
      restore screen from screl
      if keyval = "N"
        clear gets
        clear typeahead
        close databases
        return to master
      else
        retry
      endif
    endif

  else
     retry
  endif

endcase
return

* eof errorprg.prg
```


PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

This message and situation can be entirely avoided if the programmer sees that the programs request exclusive file use for every command that requires it.

An **on error** routine, if used, must be defined in every program by an **on error** command such as:

```
on error do errorprg with error(),1000,18,0,23,79
```

The parameter *timeout* in Listing 1 should be defined as public and set to a value of your choice in the top program of the system. This value will be the number of times the error routine will loop when attempting to obtain a lock.

MULTI-USER SYSTEM SETUP

When installing a multi-user system on a Novell network, all the files in the database system should be flagged as shareable for reading and writing via the network command **flag *.* srw**. Other network operating systems will usually support a similar command. If the multi-user system is being run in a single-user environment (on a single PC while not logged into the network), then the DOS **share** command should be issued each time the PC is rebooted.

Multi-user conversion is not a complicated procedure, although it does require some time and thought. It pays off, however, when your clients are able to multiply their productivity by having many people perform the same functions simultaneously and be assured of the integrity of their data. PJ

The Heap Expander™ v 2.05

Gain virtually unlimited heap space.

Function
Provides interface routines so that your programs use expanded memory, extended memory or conventional memory. As memory becomes full, The Heap Expander swaps to disk.

Utility
The Heap Expander comes with complete source code, libraries and documentation plus four utility programs that simplify development of expanded memory programs.

Guarantee
Guaranteed by The Tool Makers to perform as specified with an unlimited money-back refund.

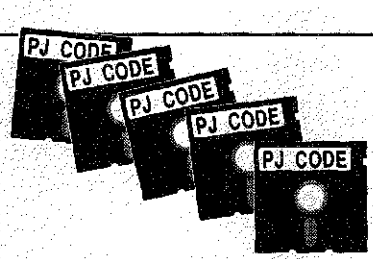
Includes libraries and source code for all current versions of Turbo C and Microsoft C.
System requirements: IBM PC, XT, AT or clone compatible with MS-DOS or PC-DOS v. 2.0 or above.

MC/VISA/COD
800-248-1045 ext 100 USA
800-952-5560 ext 100 Idaho

still just **$79.95**

THE TOOL MAKERS
P.O. Box 2151
Santa Cruz, CA 95063
408-458-0690

Circle 166 on Reader Service Card



On disk from:
Programmer's Journal
P. O. Box 70167, Eugene, OR 97401-0110
Call toll free: 800-234-0386
In Oregon call: 503-747-0800

All Code Disks $10.00 each.
To order, see ad on page 80.

PJ Code is available on the following bulletin board services:

Buffer Board
538 Cardinal Way
Springfield, OR 97477
(Data) 503-747-7636
The Buffer Board contains all of PJ's code from Sept/Oct 1987 issue through the current issue.

Generation Five Inc.
12061 Tech Road
Silver Spring, MD 20904-9998
(Voice) 301-680-3600
(Data) 301-680-9300
Accessible via PC Pursuit, Washington, D.C.
From Files submenu, select "P" for Personal Downloads.

Gilmore Systems of Southern California
5447 Indian Hills Drive
Simi Valley, CA 93063
(Voice) 805-582-1360
(Data) 805-582-9306

Ithaca Street BBS
1145 Ithaca
Boulder, CO 80303
(Voice) 303-494-8865
(Data) 303-494-8868

Mnematics Videotex
P.O. Box 19
Sparkill, NY 10976-0019
(Voice) 800-322-3633
(Voice) 914-365-0184 in N.Y.
Mention PJ when you call.

PCS-49 on CompuServe
Section 10
Sponsored by:
MISOSYS, Inc.
P.O. Box 239
Sterling, VA 22170
(Voice) 703-450-4181

Programmers Corner
5430F Lynx Ln. #224
Columbia, MD 21044
213-374-3478
(Voice) 301-740-5880
(Data) 301-596-1180
(Data) 301-997-3744

The Silicon Zone
820 Tivoli Circle #203
Deerfield Beach, FL 33441
(Voice) 305-360-9362
(Data) 305-360-9376

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

B-tree indexing

A look at indexing tradeoffs in dBASE, Clipper, and FoxPro

By M. Steven Baker

My own perspective is that one learns more from "details" than from generalities. Lots of articles and books cover material at the general level, so this article will focus on the details—how B-tree indexing is implemented in several dBASE-compatible products. My interest is not solely academic, but stems also from several years of real-world database programming conducted for the state of Oregon.

YOU WANT IT WHEN?

It all started with one of those "I need it yesterday" requirements to construct a database system on Oregon rivers. One of the program requirements was that other agencies be able to use the database files "raw" for their own purposes. We standardized on the dBASE III file format for data files, which has since proven to be a useful decision. I briefly evaluated several database programming environments. With 15,000 river segments in the system, a database engine that would provide acceptable performance even on XT-class machines with slow disk drives was needed.

All of the products I will discuss use the same database file format. The performance differences result from the indexing file data structures and the speed of the "raw code" used to implement indexing. Although all these products use B-tree indexing, each vendor incorporates some subtle differences. dBASE has its NDX index files, Clipper uses NTX, and Fox supports IDX files, none of them compatible. The differences are in the details.

A B-TREE REFRESHES

Database indexing schemes are all based on maintaining a separate index file with search keys and pointers into the actual database. Rather than searching sequentially through a large database (via the **LOCATE** command in dBASE), one matches the search key in the index file to quickly find a pointer to the database record of interest (dBASE commands **SEEK** or **FIND**). Most indexing schemes are based on tree structures for efficient searching, although extensible hashing can also be used.

Current database indexing schemes started with the discovery by Bayer and McCreight in 1970 of a new approach for external searching. The data structure, called a *B-tree*, made it possible to search and update large data files faster than existing algorithms did. Most database products, whether PC-based dBASE compatibles or mainframe SQL relational engines, now use B-tree data structures for fast indexing.

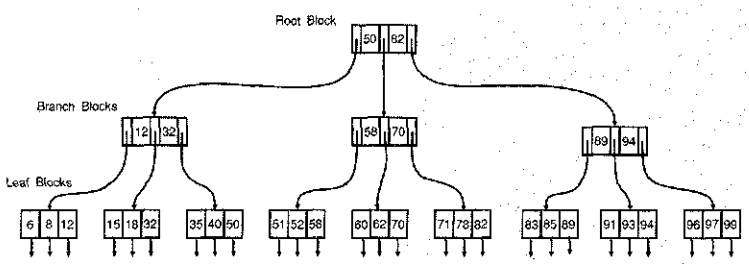
Most programmers are familiar with binary trees, which work well for searching when the search keys are in memory. But if the keys no longer fit in memory, then the speed of access is limited by the number of times one has to go to disk searching for information. Since binary tree nodes are limited to two children at the most, the height (or depth) of a binary tree can cause a large number of probes. B-trees are an extension of binary trees designed to minimize the number of disk accesses.

B-tree data structures incorporate two primary features. First, the tree structure is *balanced* so that the height of the tree is the same to any leaf node. This minimizes the worst-case search time. Second, nodes can contain *multiple keys* (multiway trees). A node in a B-tree of order *m* can contain up to *m*-1 keys and *m* pointers to children.

The database indexing structures that use B-trees are based on three types of nodes:

- a single root block node
- branch block nodes
- leaf block nodes

FIGURE 1



M. Steven Baker is the editor of Programmer's Journal *and a co-author of* Extending DOS, *published by Addison Wesley. He is occasionally found hanging around Oregon rivers.*

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

The *root* node starts the tree structure. A *leaf* is a terminal node with no children. Probably the most important rule applying to B-trees is that every node except the root must have at least *m*/2 children (see Figure 1).

The size of a B-tree node (block) is usually chosen to reflect some multiple of the disk block size. The advantage of multiple keys in a node can easily be seen in Table 1. Basically, any database system you are likely to encounter will require three to five disk probes to determine the proper database record. Once the proper node is found, a sequential search of the keys in the node will provide the record pointer into the database. Keeping the root block node in memory reduces the number of disk probes by one.

I won't go into detail concerning the techniques to generate and maintain a particular B-tree index. The vendors are of course proprietary regarding their source code. I found that the common algorithm texts (with the exception of Knuth, which is a bit dated) give surprisingly short shrift to B-trees, given their importance.

A nice feature of B-trees is that insertion is reasonably simple. If a leaf node is only partly full, inserting a record is easy—just add it to the node. When a leaf node is full, the most common scheme for inserting records splits the leaf block in half to form two nodes and inserts a new key into the original parent block. If the parent node is also full, we split the parent node and repeat this same scheme up the tree structure. If the root node is full, we simply create a new root node containing a single key. The result is that the tree structure appears to grow from the top. Deleting keys from B-trees is only slightly more complex than inserting them.

THE SIMPLE REFINEMENTS

Several refinements will improve the basic B-tree algorithms. If insertion is always done by splitting, then only about 50% of the available node space will be filled with keys (wasting valuable disk space). A modification introduced by Bayer and McCreight uses the concept of *overflow*. The goal is to reduce the temptation to split so often when inserting keys. If a node becomes over-full, before we split it we check the nearest sibling nodes for available space. This way a full node can "overflow" into a sibling node. If the sibling node is already full, both nodes can be split, forming three nodes (each about 2/3 full). Checking just one sibling node will guarantee a minimum of about 67% node utilization. In academia, these are often called B*-trees.

All the dBASE-compatible vendors use techniques to minimize splitting, as can be seen from a look at the index files they initially generate. Without overflow, their index files would be larger and sparser than necessary. A fuller B-tree index structure speeds up accesses as well. We'll take a look at the performance benefits of various refinements after dissecting the common index file formats.

TABLE 1: Maximum keys per level

	Key Entries/Node	
	10	25
Level 1	10	25
Level 2	110	650
Level 3	1,210	16,900
Level 4	13,310	439,400
Level 5	146,410	11,424,400

YE OLDE DBASE NDX FILES

The starting point of this discussion is of course the NDX files introduced with dBASE III. These database and index files are an extension of the DBF and NDX files used in dBASE II under CP/M. Although Ashton-Tate doesn't publish the details of their NDX files, a number of companies provide libraries that support DBF and NDX files. I benefitted from conversations with Ken Sawyer of Sequiter Software. Their CodeBase C library provides source code to manipulate dBASE files which can be used to develop an understanding of dBASE NDX and Clipper NTX index files. I still needed to hand-inspect NDX files and pick them apart to fully understand their structure and its implications on performance.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

LISTING 1

```
/***  dBASE III NDX file structures ***/
#define BLOCK_SIZE   512     /* our B-tree block size                    */
#define KEY_LENGTH   30      /* our example search key expression length */
#define NBYTES       2       /* filler bytes to make KEY_LENGTH mod 4    */
#define MAX_KEYS     12      /* this number varies based on key length,
          the following example is based a 30-character key expression */

typedef struct index_header {
    long    root_num;        /* block number of the root block           */
    long    eof_num;         /* block number where new block are added   */
    long    version;         /* uncertain ??                             */
    short   key_len;         /* length of the search key expression      */
    short   keys_max;        /* Maximum # of keys per block <= 42        */
    short   int_or_date;     /* Flag indicates if key is date or numeric */
    short   group_len;       /* key_len plus 8 padded to a multiple of 4 */
    short   dummy;
    short   unique;          /* Flag if index on unique keys only        */
    char    expression[100];/* Index expression used for search key      */
}  ndx_header;

typedef struct key_group    {     /* dBASE III NDX key entry             */
    long    block_num;      /* For root & branch nodes, the block number */
    long    rec_num;         /* For leaf nodes, the record number        */
    char    searchkey[KEY_LENGTH];  /* our search key                    */
    char    filler2[NBYTES]; /* filler bytes to align on 4-byte boundary*/
}   ndx_key_group;

typedef struct index_block  {
    short   num_keys;        /* number of keys in this block node */
    short   filler1          /* pad to 4-byte boundary            */
    struct  key_group[MAX_KEYS];
    long    last_block_num; /* block number if greater than key  */
}   ndx_block;
```

LISTING 2

```
/***  CLIPPER NTX file structures ***/
#define BLOCK_SIZE   1024    /* our B-tree block size                    */
#define KEY_LENGTH   30      /* our example search key expression length */
#define MAX_KEYS     24      /* this number varies based on key length,
          the following example is based a 30-character key expression */

typedef struct index_header {
    short   sign;
    short   version;
    long    root_offset;     /* offset in file of the root block         */
    long    eof_offset;      /* offset in file of free blocks            */
    short   group_len;       /* key_len plus 8                           */
    short   key_len;         /* length of the search key expression      */
    short   key_dec;         /* number of decimals in key expression     */
    short   keys_max;        /* Maximum # of keys per block;  <= 92      */
    short   keys_half;       /* Maximum # of keys per half-block         */
    char    expression[256]; /* Index expression used for search key     */
    short   unique;          /* Flag if index on unique keys only        */
}  ntx_header;

typedef struct key_group    {    /* Clipper NTX key entry               */
    long    block_offset;    /* Block offset for root & branch nodes     */
    long    rec_num;         /* For leaf nodes, the record number        */
    char    searchkey[KEY_LENGTH];  /* our search key                    */
}   ntx_key_group;

typedef struct index_block  {
    short   num_keys;            /* number of keys in this block         */
    short   key_order[MAX_KEYS]; /* sorted offsets to keys               */
    struct  key_group[MAX_KEYS];
    long    last_block_offset; /* block offset if greater than key       */
}   ntx_block
```

LISTING 3

```
/***  FoxBASE and FoxPro IDX file structures ***/
#define BLOCK_SIZE   512     /* our B-tree block size                    */
#define KEY_LENGTH   30      /* our example search key expression length */
#define MAX_KEYS     13      /* this number varies based on key length,
          the following example is based a 30-character key expression */

typedef struct index_header {
    long    root_offset;     /* offset in file of the root block         */
    long    free_ptr;        /* just a guess */
    long    eof_offset;     /* offset in file where new blocks are added */
    short   key_len;         /* length of the search key expression      */
    short   unique;          /* Flag if index on unique keys only        */
    char    expression[100];/* Index expression used for the search key  */
}  idx_header;


typedef struct key_group    {     /* FoxBASE IDX key entry               */
    union   {
        long    block_offset; /* Offset in file for root & branch nodes  */
        XLONG   rec_num;      /* In leaf nodes, record number big-endian */
    }
    char    searchkey[KEY_LENGTH];  /* our search key                    */
}   idx_key_group;

typedef struct index_block  {
    short   node_type;      /* 0 = branch node, 1 = root, 2 = leaf      */
    short   num_keys;       /* number of keys in this block node        */
    long    left_sibling;  /* offset in file to left sibling block      */
    long    right_sibling; /* offset in file to right sibling block     */
    struct  key_group[MAX_KEYS];
}   idx_block;
```

A dBASE NDX file consists of a header block followed by B-tree root, branch, and leaf blocks. The block size is fixed at 512 bytes. The header block contains the root block number, the end-of-file block number, the length of the search key, the search key expression, and a logical flag that indicates whether the index is generated only for unique keys. The dBASE header block also contains several other fields—the group length of each key entry (described below), the maximum number of keys per block, and a flag to indicate whether the key expression uses a date variable type (see Listing 1).

Root, branch, and leaf blocks may be in any order in the file. As entries are inserted, what was once a root block may become a branch block. Thus a number (a **long**) in the header block indicates the current root block in the file. Also, a second number indicates the end-of-file block where new nodes would be added. The key expression can be up to 100 characters and is stored in lowercase in the header block.

Each B-tree block contains a count (two bytes) of the number of keys stored in that block and two bytes of padding, followed by that number of search key groups. Each search key entry consists of a child node block number (**long**), a record number (**long**) in the database file, and the search key expression. The group length is the search key length plus 8 bytes for the two **long**s along with any padding needed to maintain a 4-byte boundary. The padding happens on the end of each key entry. So for a 30-character key expression (our river name example), each search key entry would be eight bytes (two **long**s) plus 30 bytes (search key length) plus two bytes of filler for a group length of 40. Blocks within the file start with block one as the first B-tree block. Block zero is the header, so a block number of zero indicates no children (a leaf block). Database record pointers are stored only in the leaf blocks. All numbers and flags are stored in Intel little-endian format.

dBASE III doesn't bother to fill new blocks with nulls before use so you'll often see garbage bytes in any unused parts of a block. This adds to the confusion when inspecting an NDX file generated by dBASE. The Ashton-Tate developers have traded file size for variable alignment. All variables in the B-tree nodes begin on 4-byte boundaries. The worst case for filler would be a key expression of just one byte over a multiple

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

of four (e.g., 29) that adds three bytes to each key entry. But most key expressions would average two bytes of filler (as in our 30-character river name, for example).

dBASE IV supports NDX index files compatible with dBASE III. dBASE IV also supports a new MDX index file format. MDX index files can contain up to 47 individual index files in a single MDX file. This allows a large number of index files associated with a single DBF file to be grouped together to free file handles under DOS. MDX files can also be specified with differing block sizes—the default block size is 1024. Only a single MDX file can be associated with a DBF file.

On inspecting MDX files, you'll find a directory structure at the beginning of the file followed by a header block and B-tree blocks for each index file entry. The basic NDX B-tree file block structure is mostly preserved inside the MDX structure, except that only a single **long** (either block number or record number) is used with each key entry. This results in smaller index files. Again, since dBASE doesn't clear out blocks with nulls before writing, MDX directory and header blocks are filled with garbage except for the few real directory entries. MDX index files are likely to provide a performance improvement over NDX files when file handles are limited.

THE CLIPPER NTX ALTERNATIVE

Nantucket Software's Clipper (Summer '87 and new version 5.0) creates NTX index files. These index files are similar to dBASE NDX files, with a few subtle changes (see Listing 2). First, the block size has been increased to 1024 bytes. This can work to an advantage since the minimum DOS cluster size on hard disks is two 512 byte sectors. With this addition, NTX files support a search key expression up to 256 characters. In some cases I suppose that a search key that long is needed, but this case looks like a poor database design to me. A 200-character search key is not going to be any speed demon, and index file sizes are going to be huge.

Clipper NTX files also add several other refinements. The NTX header block contains a pointer (offset) to a chain of free blocks created by extensive deletions. The NTX B-tree blocks also contain an array of offsets (two-bytes) at the start of each block that points to the beginning of each search key entry in the node. This array handles the order of keys within the block. When new entries are inserted into a node, the group is added at the end of existing keys; the array offsets are shuffled to reflect the proper sequential order. Both branch and leaf blocks can contain pointers to the actual database records. This should slightly reduce the size of the index files, since keys in dBASE branch blocks are also duplicated in the leaf nodes. No filler or padding bytes are used. In addition, NTX files store a *date* variable as ASCII, differing from what dBASE does.

THE FOX IDX CHALLENGER

FoxBASE and FoxPro create an IDX index file. In this case I had no help from the vendors. But, stubborn fool that I am, I spent several late nights dumping and analyzing enough IDX files (seeing patterns in my sleep) to break the puzzle. Of course, IDX files are similar to NDX files. But Fox Software developers have made some substantial refinements. The basic information is still in the header block—root node, end-of-file, key length, key expression, and unique flag. The block size is still 512 bytes. The header block adds another pointer—I assume it's to free blocks. But within the B-tree blocks are where the changes occur (see Listing 3).

IDX B-tree blocks start with a flag that indicates the kind of block—root, branch, or leaf. All of the record numbers are stored in the leaf blocks. Using the block flag, only a single **long** pointer is stored with each search key. For branch blocks, this **long** is the leaf block pointer. And within leaf blocks, each **long** is the record number of the database entry.

The IDX files also differ in that a block with 12 keys will have only 12 pointers to children—one per key expression. Child pointers always point to the block that compares "less than" to the key. The true B-tree structure has one more child pointer than it has keys in the node. Record numbers are stored as long offsets in big-endian (Motorola) format while pointers and other flags are stored in little-endian (Intel) format. As in Clipper, no filler or padding bytes are used. If a search key is 29 characters, then the group length would be 33 characters and some key groups would start on odd-byte boundaries.

Also, each leaf block contains two additional pointers to the left and right sibling blocks (with FFFFFFFFh denoting no sibling). In the academic jargon, this structure is the real B+tree. This scheme allows fast sequential scanning of a database file (the **SKIP** command in dBASE) in either ascending or descending order, without ever having to trace up and down the tree structure. IDX files trade a smaller file size and more keys per node for potential odd-byte boundaries. For example, using our 30-character river name, an IDX B-tree block will hold 13 key groups, compared to a maximum of 12 key groups in NDX files. The doubly-linked list of leaf nodes delivers very fast indexed sequential access.

Fox also stores dates and numbers in the search key in such a way that the key can be searched sequentially from left to right (big-endian). This differs from dBASE which stores dates as a floating point double (8-byte) value and numbers in Intel (little-endian) format. An interesting trade-off with the more compact index files is the possibility of key records starting on odd-byte boundaries. On older and slower 8088-based PC's, there is no penalty for odd-byte alignment. On the faster 286 and 386 machines, however, one experiences a mild penalty.

WHAT DOST THOU WANT FAST!

Well, what are the impacts of these various design decisions on indexing and access speed? Of course, this depends on what your database application is doing the most, and on which operations are more important. Some common database operations based on indexing include:

- seeking a record on a key
- inserting a record
- deleting a record
- skipping records sequentially
- creating an index from scratch

Other operations are built from these simpler operations. For example, adding a record to a database usually consists of first seeking for an existing record. If no existing record is found,

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

TABLE 2: Creating index files

	Time(secs)	Size(K)
dBASE III (1.1)-NDX	146	663
dBASE IV (1.1) -NDX	34	663
dBASE IV (1.1) -MDX	34	547
Clipper ('87) -NTX	22	626
Clipper (5.0) -NTX	30	626
FoxBASE (2.10) -IDX	30	612
FoxPro (1.02) -IDX	24	588

TABLE 3: Skipping sequentially

	Forward Time(secs)	Backward Time(secs)
dBASE III-NDX	91	350
dBASE IV -NDX	70	214
dBASE IV -MDX	43	144
Clipper87-NTX	58	537
Clipper5 -NTX	58	193
FoxBASE -IDX	15	34
FoxPro -IDX	14	25

then a new record is added and an index key entry is inserted. I include "creating" an index from scratch (which is also the same as re-indexing) as a distinct operation since database managers often have separate code to handle this case quickly (using all of the available memory).

For the Oregon river data-access system, I was most concerned with the speed to seek records and skip sequentially through files. In contrast, record addition and deletion and re-indexing happen infrequently at a central site. Other database systems will have their own unique mix of operations—viewing, adding, and deleting records.

To get a handle on performance, I wrote a series of small benchmarks to test common operations. The benchmarks perform the following simple tests:

- create an index
- seek records using an index
- skip sequentially (forward and back)
- add records
- delete records

dBASE and compatible products don't actually support deleting an index entry from a file. The database record is tagged as deleted, but still remains in an open index. Only after "packing" the database are entries deleted (at which time the file is re-indexed). The deletion benchmark is really a test of seeking and tagging the database record as deleted. In lieu of deletion, I wrote a benchmark program that changes the value of the index key—this will force an index key deletion and an insertion.

For testing, I used the same rivers database file (5MB) with almost 15,000 records. The index files were over 600K and couldn't be kept in memory. I ran the benchmarks in several environments, but the execution times reported are for an Everex 386-25 with a fast ESDI hard disk.

SPEED PAYS

Table 2 lists index file sizes and execution times for indexing on a 30-character key. The execution time for this operation is not generally crucial for an application, since it should happen very infrequently. File sizes do indicate the density of the initial index files. Substantial improvements have been made from dBASE III to dBASE IV for raw indexing, although times are the same for either dBASE IV MDX or NDX files. FoxPro improves on FoxBASE speed by about 20% and the file sizes are also smaller (less *air* in the index file). Clipper and Fox are both faster than the latest release of dBASE IV.

For sequential skipping through records, the Fox IDX indexing files are clearly the best design (see Table 3). The doubly-linked list in the IDX leaf nodes supports very fast skipping without the tree traversal required in both dBASE and Clipper. Both dBASE and Clipper display a similar anomaly and skip backward much more slowly than they skip forward.

The times for seeking records are closer together, with Fox still the fastest on this benchmark. dBASE IV MDX files are denser (28 keys per 1024-byte block) than NDX files (12 keys per 512-byte block), yet seek times are only reduced by 8%. The variation in this category is more likely a result of raw code speed and buffering.

Based on file structure, one would presume that adding records would be enhanced using Fox IDX files with their doubly-linked sibling block lists. This allows quicker access to sibling blocks without traversing the tree. The benchmark execution times shown in Table 4 indicate that Fox files are noticeably faster. Also, after adding 1000 records, the Fox index files grow in size from 60 to 100K, while dBASE and Clipper index files increase by over 250K (more air in the index).

AS THE B-TREE TURNS

These simple tests were performed on a single-user system. Some improvements possible through memory management and disk buffering can't be applied on networked multi-user databases. My own perspective is that the size of the B-tree node isn't as important as the B-tree implementation details. Fox, with its 512-byte blocks, executes very favorably compared with Clipper's 1024-byte blocks. Using large block reads (internal disk buffering), even under multi-user conditions, should mitigate much of the concern about 512-byte block sizes.

Intuition suggests that the Fox IDX file format, with its doubly-linked list of leaf blocks and higher density, is a better design. The benchmarks, though simple, clearly indicate its advantage for most operations. I'll leave the multi-user tests to the next contestant. PJ

TABLE 4: Execution times(seconds)

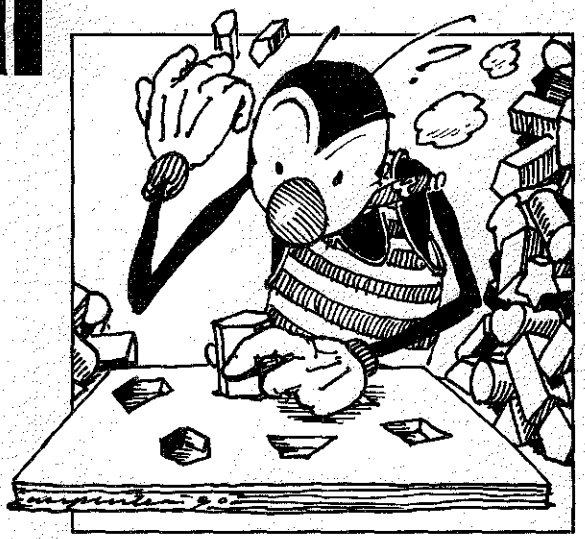
	Seeks	Adds	Changes	Deletes
dBASE III-NDX	85	1229 (916K)	408 (790K)	254
dBASE IV -NDX	50	116 (916K)	291 (790K)	138
dBASE IV -MDX	46	98 (842K)	243 (652K)	133
Clipper87-NTX	49	67 (908K)	163 (714K)	151
Clipper5 -NTX	39	73 (908K)	136 (714K)	115
FoxBASE -IDX	31	32 (677K)	101 (628K)	69
FoxPro -IDX	33	32 (695k)	99 (618K)	69

NOTE: Times are for 1,000 operations and numbers in parentheses are index file sizes after the operations.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

More virtual memory

New strategies for managing buffers and queues



By Allen Holub

In the last issue, I developed a simple disk-based virtual-memory manager for C and C++ programs. In this article, I'm going to modify the C side of my previous routines in order to improve the buffering strategy. Along the way, I'm going to discuss a pretty useful set of general-purpose queue management functions that can be used in an object-oriented way.

My previous version of the virtual memory system had one potentially undesirable characteristic. Every virtual object, as created with a **vmalloc** call, had exactly one buffer assigned to it. Sometimes this is good. It simplifies the swapping algorithm and can speed up access to the virtual object as a consequence. There are two problems, however. First of all, every object has a buffer attached to it, even if the object is not used very often in the program. This means that a lot of space can be wasted by underused buffers. The second problem is related; if you're accessing several different locations in the same virtual array and these locations are widely distributed within the array, you're going to spend your life swapping in and out of the single buffer.

Allen Holub is a teacher, writer, consultant and composer. He teaches C and compiler design for the U.C. Berkeley Extension and has authored several books, most recently Compiler Design in C *from Prentice Hall. Reach him electronically at holub@violet.berkeley.edu (...!ucbvax!violet!holub) via USENET or c/o Software Engineering Consultants, P.O. Box 5679, Berkeley, CA 94705.*

A NEW BUFFERING STRATEGY

The solution here is to allow for multiple buffers attached to a single object. Instead of attaching a buffer to every object, the virtual-memory manager controls a pool of buffers which are shared among objects. Buffers are assigned to objects on an as-needed basis, and the more heavily used objects will have more buffers assigned to them. When an object requests an array element, the memory manager determines whether the object already has that array element in a local buffer. If not, a new buffer is transferred from the pool to the requesting object and the buffer is loaded from the disk.

If no more buffers are available in the pool because all of them have been assigned to objects, the manager looks for the least-recently-used already-assigned buffer which, like Robin Hood, it steals from the associated object and gives to the new one.

You must modify the data structures for objects a little to accomplish the foregoing. Figure 1 on page 53 shows the new arrangement. The picture is pretty complicated, so I'll describe each part in turn. (I'll assume that you're already familiar with the data structures from the previous article, and only describe the changes.)

At the left of the figure is the same **hdr/vmem** pair that I used to manage the buffer last time. There are two changes here, however. First, the **dirty** bit (which indicates that a buffer has been modified since it was loaded) and the *cblock* field (which identifies the disk block which is duplicated in the buffer) have been moved from the **vmem** structure to the new **buffer** structure.

I've also changed the meaning of *cblock*: it now holds the absolute block number (the distance, in blocks, from the start of the file) rather than the relative block number (the distance from block zero) of the current object. This change makes it easier for the virtual-memory manager to steal a block from an object, because it doesn't have to chase back up the linked list to the **vmem** for that object in order to get the offset to relative block zero (which is in *hdr.offset*). Also, the *buf* pointer has been redefined from a **char*** to a **buffer***. The multiple buffers are organized as a doubly-linked list. These changes are shown in Listing 1, along with a definition for the new **buffer** structure, which is used for buffer headers.

The *next* field works like a normal linked-list pointer—it holds the address of the next **buffer** in the list. The *prev* pointer has an extra level of indirection,

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

LISTING 1

```
/*  C virtual-memory manipulation functions                         */

typedef struct hdr          /* Header for a virtual object         */
{
    unsigned    nblocks;    /* Size of array in blocks.            */
    unsigned    offset;     /* Offset (blocks) to block start.     */
    union {
        struct hdr  *h;     /* Next element of freelist.           */
        struct vmem *v;     /* Additional info if in use.          */
    } next;
} hdr;

typedef long   align;       /* worst-case alignment type.          */
typedef struct buffer       /* Information about a buffer          */
{
    unsigned cblock;        /* Absolute block # in this buffer.    */
    int      dirty;         /* Block has been modified.            */
    struct buffer  *next;   /* Pointer to next buffer in list.     */
    struct buffer **prev;   /* Pointer to the "next" field of      */
    union                   /* the previous list element.          */
    {
        align x;            /* Worst-case alignment type           */
        char  buf[BLK_SIZE]; /* The actual buffer.                 */
    }b;
} buffer;

typedef struct vmem         /* Information about virtual objects */
{
    unsigned signature;     /* 0x5a5a if structure valid           */
    unsigned ele_size;      /* size of one element                 */
    unsigned ele_per_blk;   /* number of elements in a block       */
    long     num_ele;       /* number of elements in array         */
    buffer   *buf;          /* Swap buffers                        */
} vmem;

extern buffer   *get_buffer     (vmem *obj);
extern buffer   *has_buffer     (vmem *obj,unsigned block);
extern hdr      *vmalloc        (long num_ele,unsigned int ele_size);
extern int      init_freelist   (void );
extern int      touch_buffer    (buffer *bp,int in_queue);
extern int      vclose          (void );
extern int      vfree           ( hdr *hp);
extern int      vopen           (void );
extern long     vele            ( hdr *p);
extern void     release_buffer  (buffer *bp);
extern void     *vread          ( hdr *hp,long index);
extern void     *vwrite         ( hdr *hp,long index,char *this);

static hdr      *add_vmem       (long num_ele,int ele_size, hdr *hp);
static void     *do_vread       ( hdr *hp,long index, buffer * *bp);
static unsigned enlarge         (unsigned blocks_reqd);
static int      block_io        (int read,unsigned blk_num,char *buf);
static buffer   *load_block     ( hdr *p,unsigned int req_block);
static hdr      *new_hdr        (unsigned nblocks,unsigned offset);
```

LISTING 2

```
/* Queue management definitions and macros.  (c) 1990 Allen I. Holub. */

typedef struct _queue_ele
{
   struct _queue_ele *qprev;    /* Preceding ele in queue.     */
   struct _queue_ele *qnext;    /* Next ele in queue.          */
} _queue_ele                    /* assuming worst-case ptr alignment */

typedef struct queue
{
    _queue_ele *head;   /* first ele in queue    */
    _queue_ele *tail;   /* last  ele in queue    */
} queue;

extern  void      * qmalloc   ( size_t size );
extern  void        enqueue   ( queue *q, void *usr_space);
extern  void      * dequeue   ( queue *q );
extern  int         walkqueue ( queue *q,int forw,void (*pfi)(),void *a);
extern  void      * unenqueue ( queue *q );
extern  void      * qremove   ( queue *q, void *usr_space );

#define newqueue() ((queue *)calloc( 1, sizeof(queue) ))
#define head(q)  ((q)->head ? (void *)((q)->head + 1) : NULL)
#define tail(q)  ((q)->tail ? (void *)((q)->tail + 1) : NULL)
#define qempty(q)  ((q)->head == NULL)
#define qfree(p)   free( (_queue_ele *)(p) - 1 )
```

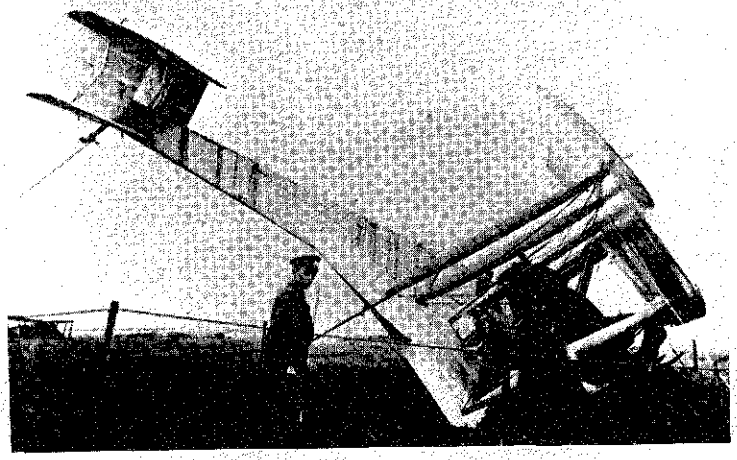
LISTING 3

```
#include <stdlib.h>
#include <stdio.h>
#include <tools/queue.h>
/* Queue.C--Queue management functions.  (c) 1990 Allen I. Holub. */
/*---------------------------------------------------------------*/
void *q_malloc( size_t size )   /* get a queue element */
{
    _queue_ele *q;
    if( q = (_queue_ele *) calloc( 1, sizeof(_queue_ele) + size ) )
        return q + 1;
    return NULL;
}
/*---------------------------------------------------------------*/
void enqueue( queue *q, void *usr_space )      /* (push) add to tail */
{
    _queue_ele *ele = (_queue_ele *)usr_space - 1;
    if( !q->head )
    {
        ele->qnext = NULL;
        ele->qprev = NULL;
        q->head    = q->tail = ele;
    }
    else
    {
        ele->qnext      = NULL;
        ele->qprev      = q->tail;
        q->tail->qnext = ele;
        q->tail         = ele;
    }
}
/*---------------------------------------------------------------*/
void *unenqueue( queue *q ) /* (pop)  remove and return ptr to tail  */
{                           /*        element or NULL if queue empty.*/
    _queue_ele *rval;
    if( rval = q->tail )
    {
        if( rval == q->head )           /* only 1 element in queue */
            q->head = q->tail = NULL;
        else
        {
            q->tail = rval->qprev;
            q->tail->qnext = NULL;
        }
    }
    return rval ? (rval + 1) : NULL ;
}
/*---------------------------------------------------------------*/
void    *dequeue( queue *q )    /* Remove and return ptr to head     */
{                               /* element or NULL if queue empty.   */
    _queue_ele *rval;
    if( rval = q->head )        /* if queue is not empty */
    {
        if( q->head == q->tail )        /* one element */
            q->head = q->tail = NULL;
        else                            /* > one element */
        {
            q->head = q->head->qnext;
            q->head->qprev = NULL;
        }
    }
    return rval ? (rval + 1) : NULL;
}
/*---------------------------------------------------------------*/
void *qremove( queue *q, void *usr_space ) /* remove arbitrary element */
{                                          /* returns 2nd argument     */
    _queue_ele *ele = (_queue_ele *)usr_space - 1 ;

    if(      ele == q->head ) dequeue   ( q );
    else if( ele == q->tail ) unenqueue ( q );
    else
    {
        ele->qprev->qnext = ele->qnext;
        ele->qnext->qprev = ele->qprev;
    }
    return usr_space;
}
/*---------------------------------------------------------------*/
walkqueue( queue *q, int forward, void (*funct)(), void *arg )
{
    _queue_ele *p;
    p = forward ? q->head : q->tail ;
    for( ; p ; p = forward ? p->qnext : p->qprev )
```

continued

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

DIRE CONSEQUENCES AWAIT USERS OF LESS MATURE TOOLKITS.



You've got to pay your dues. Before you spend a single minute, we will have spent years on our C Programmer's Toolbox. Tweaking, tinkering, making sure that you will spend less of your time getting what you want done.

Computer Language

"If I could put together a personal suite of programmer's utilities, I would include the C Programmer's Toolbox™ for a nifty Swiss Army Knife collection of utilities"

-Warren Keuffel
June 1990

Containing more than 20 tools for serious C programmers, the C Programmer's Toolbox is available in three different development environments: DOS, Macintosh® MPW, and Sun's UNIX. The Programmer's Toolbox includes support for ANSI C, Microsoft C (new support for 6.0), Turbo C, Sun UNIX C, Macintosh MPW C, THINK C (new support for 4.0 OOPs), and more. Several tools also include a C preprocessor that accepts K&R, ANSI, and other C dialects.

Handling any size program, the Toolbox is incredibly fast and extremely easy to use. Call now for more detailed information and to get your hands on our free demo disk!

C Programmer's Toolbox™	
PC Volumes I, II, & III	$125
Any 2 Volumes	$225
All 3 Volumes	$300
MPW Volume	$295
Sun Volume	$495
McTool Series	
McCPrint™	$99.95
McCLint™	$149.95

MMC AD Order Hotline:

(415)770-0858

CA residents please add sales tax.
Overseas: contact us for details. Visa and Mastercard accepted.

MMC AD SYSTEMS

The C Tool Specialists.™

The C Programmer's Toolbox, C Programmer's McTool Series, McCPrint, McCLint, and the MMC AD logo are trademarks of MMC AD Systems. All other trademarks are the properties of their respective owners.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

```
        (*funct)( p + 1, arg );
}
```

LISTING 4

```
/*  buffer management                                           */
#include <signal.h>
#include <tools/queue.h>

#define BUFFERS 40                  /* Number of buffers.       */
static  buffer *Free_bufs = NULL; /* Available buffers.         */
static  queue  *Active    = NULL; /* Queue of active buffers. Least */
                                  /* recently used buffer at head.  */
/*-------------------------------------------------------------*/
int init_freelist()               /* Allocate all the buffers and   */
{                                 /* put them into the free list.   */
    buffer *b;                    /* Return false on failure.       */
    int     i;

    for( i = BUFFERS; --i >= 0; )
    {
        if( !(b = (buffer *) q_malloc(sizeof(buffer)) ))
            return 0;
        b->next   = Free_bufs;   /* add new buffer to head of list */
        Free_bufs = b;
    }

    return 1;
}
/*-------------------------------------------------------------*/
buffer *get_buffer( vmem *obj )
{
    /* Get a new buffer for the virtual-memory manager and return a */
    /* pointer to it. Note that a stolen buffer is removed from     */
    /* the queue. The calling function should update the queue with */
    /* a call to touch_buffer(). get_buffer() initializes the       */
    /* Active queue and freelist the first time it's called.        */

    buffer *b;

    if( !Active && !( (Active=newqueue()) && init_freelist() ))
    {
        raise( SIGABRT );  /* Try to abort the program.       */
        return NULL;       /* If SIGABRT returns, return NULL */
    }
    if( Free_bufs )               /* If there's a buffer in */
    {                             /* the free list, use it; */
        b = Free_bufs;
        b->dirty  = 0;            /* suppress write before read */
        Free_bufs = b->next ;
    }
    else                          /* otherwise, steal one   */
    {                             /* from an object.        */
        b = dequeue( Active );       /* Get least-used buf.*/
        if( *(b->prev) = b->next )   /* remove it from     */
            b->next->prev = b->prev; /* the object.        */
    }
    return b;
}
/*-------------------------------------------------------------*/
touch_buffer( buffer *bp, int in_queue )
{
    if( !in_queue )
        enqueue( Active, bp );
    else if( tail(Active) != bp )        /* if not already at tail */
        enqueue( Active, qremove( Active, bp ) );   /* move to end */
}
/*-------------------------------------------------------------*/
void release_buffer( buffer *bp )       /* release the buffer */
{
    /* It's an error if *bp is already in the free list. It must  */
    /* be an active buffer, enqueued in the Active queue, and not */
    /* in the free list.                                          */

    qremove( Active, bp );       /* Pull it out of the active list */
    bp->next = Free_bufs;        /* and move it to the free list   */
    Free_bufs = bp;
}
```

LISTING 5

```
/*  Virt.c-modified to support multiple buffers                 */

buffer *has_buffer( vmem *obj, unsigned abs_block )
{
    /* If the indicated object already owns a buffer that holds    */
    /* the indicated block, return a pointer to that buffer,       */
    /* otherwise return NULL.                                      */

    buffer *bp;
    for( bp = obj->buf; bp ; bp = bp->next )
        if( bp->cblock == abs_block )
            break;
    return bp;
}
/*-------------------------------------------------------------*/
PRIVATE hdr *add_vmem( long num_ele, int ele_size, hdr *header )
{
    vmem *p;
    if( !(p = (vmem *)malloc( sizeof(vmem)) ) )
    {
        errno = ENOMEM;
        return NULL;
    }
    else
    {
        header->next.v = p;
        p->buf         = NULL;          /* Initially, no buffers */
        p->signature   = SIGNATURE;
        p->ele_size    = ele_size;
        p->ele_per_blk = BLK_SIZE / ele_size;
        p->num_ele     = num_ele;
        return header;
    }
}
/*-------------------------------------------------------------*/
PUBLIC int vfree( hdr *hp )
{
    int    rval     = 1;
    hdr    *cur, *h;
    vmem   *v       = hp->next.v;
    buffer *bp, *next;

    if( v->signature == SIGNATURE )
        v->signature = 0;
    else
    {
        errno = EBADF;
```

continued

however. It points at the *next* field in the previous header, not at the previous **buffer**. The extra level of indirection means that the first header in the list is not a special case. That is, if the *prev* field was a pointer to a **buffer**, you'd have to put an entire **buffer** into the **vmem** structure in order to be able to delete the buffer at the head of the list without an unacceptable amount of extra code. This would, of course, waste a lot of memory because you only need a single head-of-list pointer in the **vmem** itself. The current arrangement lets you delete a node that has been deleted from the list without regard for the position of the node in the list. For example, given:

```
buffer *p;  /* Pointer to an arbi- */
            /* trary list element. */
```

you can remove the node pointed to by *p* from the list by means of the following bit of code:

```
if( *(p->prev) = p->next )
    p->next->prev = p->prev;
```

The assignment in the **if** statement adjusts the *next* field of the previous node to point around the current node. Then, if the current node wasn't the last one in the list, the *prev* pointer of the next node is adjusted to point back at the current node's predecessor. Note that this code works whether or not the current node is the first one in the list.

The declaration of the actual buffer (*buf*) within the **buffer** structure also needs some explanation. The actual buffer must satisfy the worst-case alignment

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

restriction on the machine in order to make these routines as general-purpose as possible. You must be able to put any sort of object into the buffer without regard to alignment.

This situation is similar to that of the pointer returned from **malloc**, which must also satisfy worst-case alignment restrictions because **malloc** has no idea what the returned memory is going to be used for. A practical example: if a virtual array of **longs** is being implemented, you must be able to say something like this:

```
buffer swap_buffer;
((long *)(swap_buffer.b.buf))[i]
```

to get an array element. This means that the first element of the array must be physically placed at the beginning of the buffer. You can't just declare the buffer as a simple array of **char** because the alignment restrictions on a **char** are usually looser than those for a **long**

Turn your favorite C compiler into a powerful database manager with the

C/Database Toolchest

The **C/Database Toolchest™** adds sophisticated file management functions to your Power C™, Turbo C®, QuickC®, or Microsoft® C compiler. With the **C/Database Toolchest™**, your data requires much less disk space than with programs like dBASE®, and you can access your data much faster. Of course the full power of C provides you with an unlimited amount of programming flexibility.

The **C/Database Toolchest™** includes three major components:

1) An advanced B+tree library gives you instant access to your data.

2) A high-level ISAM library provides you with an easy-to-use C interface, and

3) A complete database manager (with C source code included) shows you how to create impressive applications.

You also receive a comprehensive 350 page manual and a utility for converting dBASE® files.

The **C/Database Toolchest™** supports features that you'd expect to find only in products costing ten times as much. Advanced features include variable length records, variable length keys, multiple keys per index, and multiple indexes stored in a single file. Your data files can contain an unlimited number of records, and each record can be as large as 32K bytes in length.

About the only thing that the **C/Database Toolchest™** doesn't do is cost you a lot of money. We've kept our price low so you can manage your budget as easily as your data.

Now Only $19.95!

Order Coupon

Name_____
Street_____
City_____
State_____ Zip_____
Telephone_____
Paying By___ Money Order___ Check
___ Visa ___ MC ___ AX ___ Disc.
Card#_____
Exp. Date_____
Disk Size___ 5¼" ___ 3½"

Qty	Product	Price	Subtotal
___	C/Database Toolchest	$19.95	_____
___	C/Database Library Source, B+tree & ISAM library source code	$10.00	_____
	Add Shipping ($5 USA, $20 Foreign)		_____
	Texas Residents Add 8% Sales Tax		_____
	Total Amount of Your Order		_____

Order now by calling our toll free number or mail the coupon to:

Mix Software
1132 Commerce Drive
Richardson, TX 75081

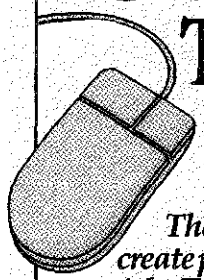
1-800-333-0330

60 Day Money Back Guarantee
Not Copy Protected ■ Royalty Free
For technical support, please call:
1-214-783-6001

C/Database Toolchest and Power C are trademarks of Mix Software. QuickC and Microsoft C are registered trademarks of Microsoft. Turbo C is a registered trademark of Borland. dBASE is a registered trademark of Ashton-Tate.

Circle **266** on Reader Service Card

UNLEASH TURBO SPEED



The fastest way to create powerful programs with Turbo Pascal 5.5, Turbo C 2.0 and Turbo C++!

POWER TOOLS PLUS™/5.1 – $149

– lightning-fast routines to help you:

◆ Add easy-to-use integrated mouse support for windows and menus ◆ Generate context sensitive help screens ◆ Resize and move windows and use drop shadows for that professional look ◆ Let users choose from window-oriented pick lists ◆ Create and access "huge" data structures ◆ Use multiple-line edit fields with fully configurable edit keys ◆ Add EMS support ◆ Write TSRs and ISRs easily ◆ Create powerful programs in Turbo Pascal 4.0, 5.0 & 5.5!

Turbo C TOOLS™/2.0 – $149

– fast, high quality functions to help you:

◆ Add easy-to-use integrated mouse support for windows and menus ◆ Quickly include virtual windows and menus ◆ Integrate your windows and menus with Turbo C's text windows ◆ Create context-sensitive help screens ◆ Provide multiple-line edit fields with fully configurable edit keys ◆ Write TSRs and ISRs easily ◆ Create powerful programs in Turbo C 1.0, 1.5, 2.0 and Turbo C++!

FREE with these products!

All source code, complete sample programs, and a comprehensive manual are included. We offer free technical support and a bulletin board dedicated to technical issues.

Unleash your potential!

We offer programming tools that are fast, flexible and affordable. Call now to order, or to ask for a free brochure on our full line of products for C and Pascal.

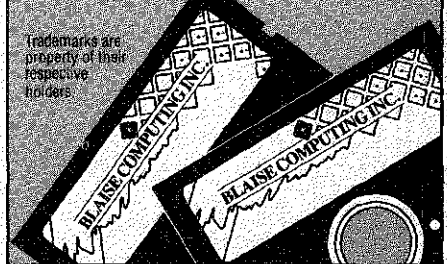
Put Blaise tools to the test!

If during the first 30 days you are not satisfied, we'll refund your money.

Call (800) 333-8087 today!

BLAISE COMPUTING INC.
2560 Ninth Street, Suite 316 Berkeley, CA 94710
(415) 540-5441 FAX (415) 540-1938

Trademarks are property of their respective holders



Circle **14** on Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

A professional workstation environment for OS/2®

Hamilton C shell™

"...much more powerful than CMD.EXE...blindingly fast... we have a winner...a much-needed and well-done product."
–Personal Workstation Magazine

The superior alternative to the standard OS/2 command processor. Faithfully recreates the entire C shell language as described in the Berkeley 4.3 *UNIX® Programmer's Manual*. Created explicitly for OS/2. Not one line ported from or created on anything but OS/2. Extensively multithreaded.

Features: Command line editing • History • Filename and command completion • Arrow and function keys • Enormous 64KByte command lines • Aliases and shell procedures • PATH hashing • Recursive filename wildcarding • Fully nestable control structures • Powerful expression grammar • Command substitution • Background threads and processes.

Over 113 commands: `alias, cat, chmod, cls, cp, cut, diff, dirs, du, eval, fgrep, grep, hashstat, head, history, label, ls, kill, markexe, more, mv, popd, ps, pushd, pwd, rm, sed, sleep, split, strings, tabs, tail, tar, tee, time, touch, tr, uniq, vol, wait, wc` and others.

Supports HPFS and long filenames.

Requires OS/2 1.1 or later. All executables will run properly in a Presentation Manager window. Not copy-protected.

$350.00. Unconditional satisfaction guarantee. MasterCard & Visa accepted. ($365.00 Canada/Mexico; $395.00 elsewhere.)

Hamilton Laboratories
13 Old Farm Road, Wayland, MA 01778–3117, U.S.A.
Phone 508-358-5715 • FAX 508-358-1113 • BIX hamilton

Circle 334 on Reader Service Card

A Difference That Must be Seen!

Diff_it™

The difference program that you've been waiting for, you've seen the others, now see the difference.

- △ *Unlimited file sizes.*
- △ *Scrollable split screen display.*
- △ *Display in ascii or hex.*
- △ *Configurable compare algorithm.*
- △ *25, 43 and 50 line display modes.*
- △ *Command line interface.*
- △ *Pop-up menu interface.*
- △ *Pop-up help screens.*
- △ *Difference output to disk file.*
- △ *Fast Terse modes.*
- △ *Line, Word and Hex compare.*
- △ *Batch File drive-able.*

Requires IBM PC or 100% compatible, 148KB memory, DOS 2.0 or higher, floppy or hard disk. Not copy protected. 30 day money back guarantee, send check or money order. $129.95

ORDER NOW!

DOUBLE R SOFTWARE, INC.
RD 1, Fifth Ave, Valley Forge Manor
Phoenixville, PA 19460
(215) 666-6104

Circle 31 on Reader Service Card

Complete Modem Communications

TURBO PASCAL & C programmers

Introducing
Sugar Coated Serial™

A complete library of Turbo Pascal & C procedures, functions and low-level assembly routines that allows you to EASILY incorporate MODEM communications into your application.

Features:

- All source code included (including Assembler).
- No knowledge of assembler required.
- No royalties or fees for use in your programs.
- Easily do XMODEM, YMODEM, & Telink file transfers (including multiple files with YMODEM.)
- Pull-down menu communications program included using only routines from this library.
- Complete user manual including many examples.
- Much more...call for the information pak to get the full details.

For IBM PC/XT/AT compatibles using T.P v5.0 or >

* Complete System:	$109.95
* Demo Disk:	$ 5.95
* Information Pak:	Free

Shipping included, IL residents add 6.25% sales tax. Specify disk size.

M C
V i s a

Send Check, M.O. or Credit Card info with order to:

Goin Places Software
Suite 145
1202 N West 75th Street
Downers Grove IL 60516

For phone orders or info call: (708) 759-1270

Circle 18 on Reader Service Card

(which often must have an address that is an even multiple of four—the **char** can be anywhere in memory). The problem is solved by putting the buffer (*buf*) together in a union with a worst-case alignment object. The *x* field of the union is never used—it just guarantees that the first character of the buffer will satisfy the alignment restrictions of the **align** type.

BUFFER MANAGEMENT

Referring again to Figure 1, the buffers that are currently in use are also linked together into a second linked list, represented by the boxes beneath each **buffer**. This second linked list is actually a queue, and it's used for buffer management. When the virtual-read or write function needs to access a particular array element, it chases down the linked list of buffers looking for one that has the correct block in it. (The block number is in the **buffer**'s *cblock* field.) If there is no such buffer, the virtual-memory function must get a new buffer from the buffer manager and load that buffer from the disk. The buffer manager keeps a linked list of available buffers around and, if it can, it will give the requesting object a new buffer from that linked list.

If there are no available buffers, however, the buffer manager resorts to sterner measures—it must take a buffer away from a virtual object and give that buffer to the requesting object. There are several strategies that can be used to decide which buffer to decommission, but the one used here is a "least recently used" strategy, implemented with the queue I just mentioned.

Every time that a buffer is accessed, the buffer manager removes that buffer from its current position in the queue and then enqueues that buffer back at the tail end of the queue. This way, the buffers that are used least often will migrate toward the front of the queue. So when an object requests an empty buffer and one isn't available, the buffer manager just takes the one off the head of the queue and gives it to the requesting object. (It must also modify the buffer chain in the object to detach that buffer from the object.)

You could avoid using a queue by time-stamping all of the buffers, but then you'd have to search the whole list

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

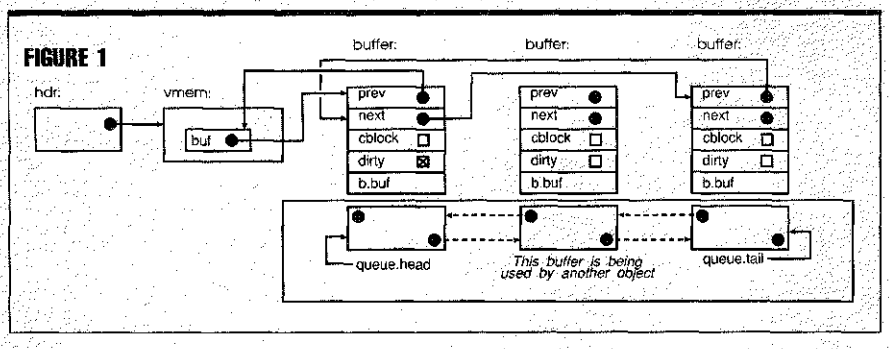


FIGURE 1

of them to find the one with the oldest time-stamp—a time-consuming process at best. A queue lets you manipulate the buffers faster, at the cost of a few additional pointers in each buffer.

Note that this mechanism can also work to your advantage when all the buffers are assigned to the same object. The buffer manager will effectively take the least-used buffer from the object and then give it back, but this time the buffer will be at the head of the linked list of buffers so it can be accessed more quickly.

QUEUE MANAGEMENT

The queue management is implemented here using an object-oriented design, although the objects are implemented in C, not in C++. Listings 1 and 2 contain a group of general-purpose queue management functions and macros. A queue is declared and initialized as in the following code:

```
#include "queue.h"
queue  *q;
if( !(q = newqueue()) )
    printf("Out of memory");
```

It's deleted with a simple **free** call:

```
free(q); /* Done with queue */
         /* discard it.     */
```

Once the queue is created, an arbitrary object that's destined to be enqueued is allocated as follows:

```
struct object { ... } *mp ;

if( !(mp = q_malloc(sizeof(object)) ))
    printf("Out of memory");
```

The **q_malloc** function is used just like **malloc**—you pass it an object's size and it returns memory that can be used for the object. You would initialize the object in the normal way and enqueue it (at the tail of the queue) with:

```
enqueue( q, mp );
```

You can remove an object from the head of the queue like this:

```
if( !(mp = dequeue(q)) )
    printf("queue is empty");
```

This function returns **NULL** if the queue is empty; otherwise it removes the head-of-queue element and returns a pointer to it. The memory for the queue element can be discarded with the following call:

```
qfree( mp );
```

Be very careful not to use **free** for this purpose. The pointer returned from **qmalloc** is not the one that was returned from **malloc**, so a direct call to **free** here will corrupt the memory pool. You don't have to discard the memory, though. You can reinitialize it and then re-enqueue it with a second **enqueue** call.

Two other functions are provided for removing objects from the queue. A call that looks like:

```
if( !(mp = unenqueue( q )))
    printf("queue is empty");
```

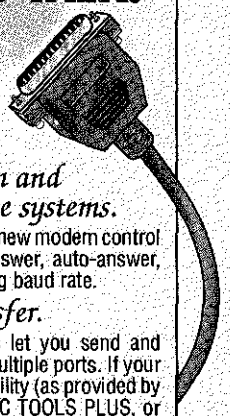
lets you use the queue as a stack. It removes the object that is currently at the tail of the queue (the most-recently-enqueued object) rather than the one at the beginning. You can look at **enqueue** as a push operation and **unenqueue** as a pop. Also of interest, particularly in the current application, is the **qremove** function:

```
qremove( q, mp );
```

This function is passed (in the second argument) a pointer to an object that's already in the queue. It removes that object from the queue and returns a pointer to it. The pointer is the same thing as the incoming second argument, but it's occasionally useful. A few other utility functions and macros are also provided.

WE HAVE WAYS TO MAKE YOU TALK.

(And let you listen in, too)



with tools from Blaise Computing.

Talk to your modem and connect with remote systems.

Go online with ease with our new modem control routines. Initialize, dial, answer, auto-answer, and auto-adjust for incoming baud rate.

Let's talk file transfer.

XModem/YModem routines let you send and receive multiple files over multiple ports. If your program has ISR/TSR capability (as provided by our POWER TOOLS PLUS, C TOOLS PLUS, or Turbo C TOOLS), you can talk file transfer in the background while your foreground process talks data input, or database query, or whatever.

The file transfer capabilities include 1K packets, CRC error checking, YModem (multi-file transfers with file name and size preserved), automatic adaptation to incoming packet size and error detection method. And we're talking fast — all file transfer routines have been optimized for maximum throughput.

Let's talk basics.

C ASYNCH MANAGER™ and **ASYNCH PLUS™** retain the features which have made them the libraries of choice for asynch projects — buffered interrupt-driven input and output to multiple COM ports with speeds up to 19200 Baud, XON/XOFF protocol, hardware handshaking and much more.

C ASYNCH MANAGER supports Microsoft C, QuickC and Turbo C. ASYNCH PLUS supports Turbo Pascal and QuickPascal. Both products require DOS 2.00 or later and an IBM PC, XT, AT or PS/2 machine or true compatible.

Let's talk price.

Why pay more? Our ASYNCH packages are just $189, including complete source code, a comprehensive reference manual with extensive examples, sample programs and online help.

Turn that serial cable into a party line with View232!

Debugging serial communications doesn't have to leave you talking to yourself. **View232™** turns your computer into a serial data analyzer that lets you listen in on the conversation between any two serial devices — and this party line provides a transcript of what was said!

View232 displays the data as it flows in both directions. Save a whole transmission in a buffer, then browse through or search the buffer for a pattern or specific character. And you can save the data to disk or print it for later study.

VIEW232 is easy to use. And we supply the cable, all for just $189!

Let's talk track record.

Blaise Computing has produced a collection of tools that are unsurpassed for reliability, flexibility, and ease of use.

And, we're talking guaranteed.

If during the first 30 days you're not completely satisfied, we'll refund your money.

Call (800) 333-8087 today!

BLAISE COMPUTING INC.

2560 Ninth Street, Suite 216
Berkeley, CA 94710
(415) 540-5441
FAX (415) 540-1938

Trademarks are property of their respective holders.

Circle 2 on Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

NEW V 4.1

Q PRO 4 TM

lean, mean, programming machine

WORLD'S FASTEST!
MOST COMPACT!

4GL FOR APPLICATIONS DEVELOPMENT PROS

Q-PRO 4 is the best 4GL for real professional applications developers. This compact, professional quality, rock solid interpreter is field proven with over **25,000** installed. For DOS/LANs/ PCMOS-386 with full record and file lock – Novell and NetBios! ConDOS version native mode (.CMD files & code sharing – up to 4 users in 640k!).

Blazing File and Screen Speed!

Q-PRO 4's speed rivals compiled C. Runs nicely on a 4.77 mhz XT and flies on an AT! For example:

6 min. – Write 10,000 unique 255 byte records and update 4 indices. Using this file:

24 secs. – Add 100 records at random locations.

0.2 secs. – Position to a record by its primary index; display the record's 50 fields formatted on the screen.

Amazingly Compact

The entire Q-PRO 4 interpreter is less than 50k bytes of raw power! Your whole application will need less than 300k of RAM. You won't need memory stretchers or other DOS workaround gimmicks!

Its complete 115 command syntax and advanced 4GL event driven architecture speed your development. Easily does 1-to-1, 1-to-many and many-to-many relationships. Extensible -- add your own routines in C or assembler! Optional distribution system encrypts and locks your code for safe distribution. No royalties!

Q-PRO 4 includes: unique multiple table driven data entry screens with full color control; dialog boxes; light-bar menus; selection windows; data dictionary; report Generator; query language (optional).Can your DBMS do this?

3 Commands – Retrieve a record using its key and display all its fields to the screen.

2 Commands – Write a full screen (up to 1900 fields) to an indexed file and update all keys.

NO Commands – Build a complete data entry screen with fill-in-the-blanks entry fields, prompts and field edits (justification, type checking, must tab, etc.).

Order Q-PRO 4 now! Half price introductory offer: $200.00 (+7.75 S&H) until 12/31/90. 30 day guarantee. Check, VISA, MC, COD. Demo/eval systems available.

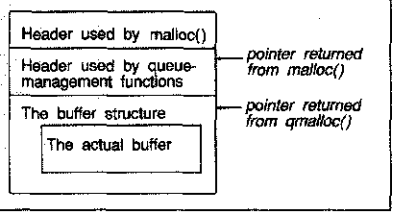
QNE International®
136 Granite Hill Court Langhorne, PA 19047
TEL: (800) 333-0448 FAX/MODEM: (215) 968-3550
(All timings on 8mhz AT Clone, 30 ms disk.)

```
        rval  = 0;
        goto end;
    }

    for( bp = v->buf ; bp ; bp = next )    /* NEW CODE:                */
    {                                      /* Release all the buffers */
        next = bp->next;                   /* owned by the current    */
                                           /* object                  */
        if(bp->dirty && !block_io(WRITE, bp->cblock, bp->b.buf))
        {
            rval = 0; /* errno set from read/write call in block_io */
            goto end;
        }
        release_buffer( bp );
    }
    free( v );          /* discard the vmem component   */

    if( !Freelist )     /* Free list is empty. Create a new one. */
    {
        Freelist = hp->next.h = hp;
        goto end;                          /* return default value */
    }
    for( cur = Freelist ; 1 ; cur = cur->next.h )
    {
       if(  ( cur->offset < hp->offset &&
                            hp->offset < cur->next.h->offset
            ) ||
            ( cur->offset >= cur->next.h->offset &&   /* 1 */
                (cur->offset < hp->offset             /* 2 */
                   || hp->offset < cur->next.h->offset /* 3 */
                )
            )
         )
            break;
    }
    if( cur == cur->next.h )    /* only block in chain */
    {
        if( hp->offset + hp->nblocks == cur->offset )
        {
            cur->offset   = hp->offset;
            cur->nblocks += hp->nblocks;
            free( hp );
            goto end;
        }
        else if( cur->offset + cur->nblocks == hp->offset )
        {
            cur->nblocks += hp->nblocks;
            free( hp );
            goto end;
        }
    }
    if( hp->offset + hp->nblocks != cur->next.h->offset )
        hp->next.h = cur->next.h;
    else
    {
        hp->nblocks += cur->next.h->nblocks;
        hp->next.h   = cur->next.h->next.h;
        free( cur->next.h );
    }

    if( cur->offset + cur->nblocks != hp->offset )
        cur->next.h = hp;
    else
    {
        cur->nblocks += hp->nblocks;       /* Adjacent */
        cur->next.h   = hp->next.h;
        free( hp );
    }
end:
    return rval;
}
/*-------------------------------------------------------------*/
PUBLIC void *vread ( hdr *hp, long index )
{
    /* Note that the returned pointer is now good only until */
    /* the next virtual-memory-access function is called.    */

    buffer *junk;
    return do_vread( hp, index, &junk );
}
PRIVATE void *do_vread( hdr *hp, long index, buffer **bp )
{
    /* Workhorse used by vread. Works like vread used to work, but  */
    /* modifies *bp to point at the buffer that holds the cell.     */
```

continued

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

FIGURE 2



The first is:

```
walkqueue( queue *q, int forward,
        void (*f)(), void *arg )
```

which walks through the queue from head to tail (if the *forward* argument is true) or from tail to head (if *forward* is false). It calls the function pointed to by *f* for every queue element in the following manner:

```
(*f)( ptr_to_queue_element, arg );
```

where **arg** is the fourth argument to **walkqueue**. This function would typically print the element's contents or look for some arbitrary information in the element. Three other functions are useful: **head(q)** returns a pointer to the object currently at the head of the queue (or **NULL** if the queue is empty); **tail(q)** returns a pointer to the object at the end of the queue (or **NULL**); and the function **qempty(q)** returns true if the indicated queue is empty.

The implementation of these functions is straightforward, although they involve a lot of heavy duty shuffling around of pointers. Take a look at Listings 1 and 2 if you would like to see more details.

THE QUEUE FUNCTIONS

The queue functions are structured in an object-oriented fashion: the mechanics of queue manipulation are entirely hidden from the application program. The application creates a high-level object—a queue—with a **newqueue** call, and then manipulates that object with various subroutine calls. It doesn't know (or care) how those functions go about doing their work as long as the functions behave in a reasonable way.

This sort of organization really pays off in an application like the current one, because the buffer-management functions don't need to worry about the mechanics of queue management. You should note that the **buffer** definition back in Listing 1 doesn't declare any pointers that are used for queue management; rather, **buffer**s are allocated as follows:

```
buffer *p;
p = qmalloc(sizeof(buffer))
```

The buffer functions don't have to worry about the queue management stuff; they just use the queue management functions for that purpose.

If you were doing this in C++, the queue management functions would all be encapsulated in (made members of) the **queue** structure. A **buffer** class would then be derived from the **queue** base class like this:

```
class buffer : private queue {...}
```

Note that this is a **private** base class (which is the default, but it's a good idea to put the keyword in there for documentation purposes) because a **buffer** isn't really a refinement of a **queue** object—it just shares some of the properties of a **queue**. There's no reason to access queue functions through a pointer to a **buffer**. Also note that the class derived from C++ organizes memory exactly as it is organized here. There are three layers of headers at work, one for each derivation level. (I'm looking at the **queue** as derived from a **malloc** base class.) The situation is shown in Figure 2 above.

All that's left to do is to modify the virtual-memory functions so as to support the new buffering scheme. The first order of business is providing a few buffer management functions (as shown in Listing 1) which you should read now. Note that I've continued to use an object-oriented methodology with the new subroutines. All of the queue management is done by the buffer management functions. The virtual-memory manager doesn't even know by what process the buffer manager decides how buffers are assigned. It simply requests a new buffer and gets a pointer back to the new buffer.

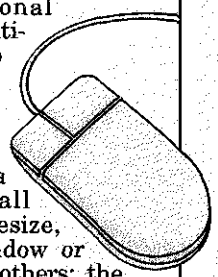
Our next task is to modify the virtual-memory functions from my last article. The **block_io**, **vele**, **vopen**, **vclose**, **vmalloc**, **new_hdr**, and **enlarge** functions don't need to be changed. The **vdirty** function is no longer safe to use because the buffer used for the initial read or write might be recycled by the

6.0 AND COUNTING!

Integrate sophisticated features into your Microsoft C and QuickC applications with

C TOOLS PLUS/6.0™

C TOOLS PLUS version 6.0 is filled with many advanced routines for developing high-powered C applications, including: virtual, stackable menus and windows with full mouse support and optional "drop shadows"; multiple virtual pop-up help screens; a miniature multi-line editor for gathering user responses in a robust fashion; a single function call which can move, resize, and promote a window or menu on top of all others; the ability to update covered windows automatically when they are written to; support for EGA, VGA, and MCGA text modes including 30-, 43-, and 50-line modes; support for the enhanced (101/102 key) keyboard.



All this and more for only $149!

C TOOLS PLUS/6.0 also contains functions for writing interrupt service routines; creating pop-up memory resident applications; general memory "peeks" and "pokes"; access to the DOS PRINT utility; as well as many other general utility functions and macros.

COMPLETE PROFESSIONAL PACKAGE.

Blaise Computing's function libraries offer easy to use solutions to your programming needs. You get source code, complete sample programs, and a comprehensive reference manual with extensive examples. Supports QuickC and Microsoft C 5.0 and later.

30 DAY GUARANTEE.

If during the first 30 days you are not completely satisfied, we'll refund your money.

Other powerful products from Blaise Computing

Circle 15 on Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Program FASTER with ...

CC-RIDER

or the new C++ RIDER

POP-UP SOURCE BROWSERS FOR ANY EDITOR !

CROSS-REF Editing!

Instantly pop-up your symbol definitions with a *hypertext hot-key* inside any editor! The *Standard Edition* has Source View, Edit and Paste commands. And the powerful *Professional Edition* even lets you walk through all *uses* of a symbol in your code as you edit ! C++ RIDER fully supports C++ v2.1, including classes, constructors, overloading, etc.

Source Analysis!

Build a database of symbol information with our powerful ***source code analyzer*** which provides a wealth of useful output, like fully commented static and global function prototype #include files and a detailed log file showing WHERE AND HOW all your symbols are used. And it now uses expanded memory for ***lightning fast*** source code scanning!

QuickHelp, too!

CCSYM can also create a ***help database for your program*** which is compatible with Microsoft's help system, *QuickHelp*. All symbols in your application are accessible from QuickHelp's menus, with ***documentation extracted automatically from your original source code.*** Works with ANSI/Microsoft C, plus AT&T/Zortech/ Borland C++.

Western Wares

For DOS and OS/2

$89 / 179 / 279

(303) 327-4898

Box C Norwood, CO 81423

FREE DEMO DISK

Circle 100 on Reader Service Card

4c™ Hypertext Source Code Databank for Programmers

Explore source code! Navigate uncharted logic trails easily!

4 c includes:

★ A Source Code Analyzer:
The analyzer maps and cross-references functions and variables to their respective path and file names. The cross-reference table can be displayed alphabetically and by type of variable. It allows a programmer to edit functions directly without specifying a file name.

★ A Point-and-Shoot Zoom Key:
Which instantly locates any function, global variable, structure definition, #define statement, or macro, etc., in any file, directory, and drive, and displays it in a new edit window.

★ A Full Featured Built in Editor:
Featuring multiple files, multiple windows, cut-and-paste between files, search-and-replace, auto indent, tab support, and a familiar user interface with pull-down menus, or use your own editor!

"Given a procedure's identifier, an editor should be able to bring up the full procedure interface, including parameters, so that you don't have to remember how many and in what order. Wouldn't it be nice to hit a hot key on the name of a complicated data type and have its definition pop up in a window? It may seem like a real small matter, but how many of you still flip through printouts doing things the editor should be doing?"

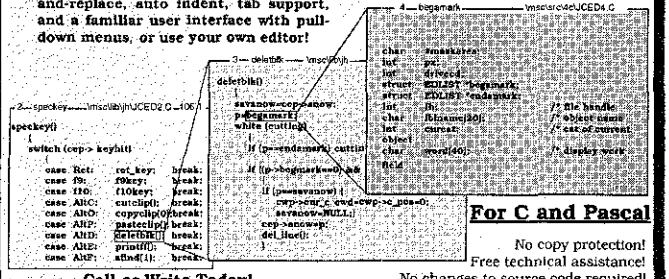
- Phillipe Kahn, "The Connection", Spring/Summer 1988

4 C does this and more

"4c's Magic is apparent..."
- PC Magazine, February 1989

"Daylight in the C Confusion"
- Toolbox magazine, December 1988

"...Destined to become the standard editor for C..."
- Computer Personlich magazine, October 1988



For C and Pascal

No copy protection!
Free technical assistance!
No changes to source code required!
Required IBM PC compatible with 350K RAM.
30 Day Money Back Guarantee!
Only $119.00 plus shipping
Visa, Mastercard, check, money order, or COD.

Call or Write Today!

Tri-Technology Systems Inc.
1225 S. Elgin, Forest Park, IL 60130

Phone: 708-366-7595

Circle 276 on Reader Service Card


```
    buffer   *b;              /* NEW: buffer that holds the cell.   */
    vmem     *v = hp->next.v; /* vmem componant of current object.  */
    unsigned rel_block_num;   /* Relative block number. (ie. from   */
                              /* start of object, not start of file)*/

    if( index < 0  ||  index > v->num_ele )
    {
        errno = ERANGE;       /* index is out of range */
        return 0;
    }
    if( v->signature != SIGNATURE || Vfd == -1 )
    {
        errno = EBADF ;       /* signature doesn't match */
        return 0;
    }

    rel_block_num = index / v->ele_per_blk ;
    if( !(b = load_block( hp, rel_block_num + hp->offset )))
        return NULL;
    *bp = b;

    index -= rel_block_num * v->ele_per_blk;    /* (1) */
    index *= v->ele_size;                       /* (2) */
    return( b->b.buf + index );                 /* (3) */
}
/*-------------------------------------------------------------*/
PUBLIC void *vwrite( hdr *hp, long index, char *this )
{
    /* Transfer *this to the virtual array at the indicated index. */
     * Return a pointer to the actual memory location for *this    */
     * or NULL on an i/o failure or a bad "hp" argument. Note that */
     * the returned pointer is good only until the next virtual-   */
     * memory access function is called.                           */

    int    i;
    char   *ele_p;
    void   *startp;
    buffer *bp;
    vmem   *v     = hp->next.v ;

    if( startp = ele_p = do_vread( hp, index, &bp ) )
    {
        bp->dirty = 1;
        for( i = v->ele_size; --i >= 0 ;)
            *ele_p++ = *this++;
    }
    return startp;
}
/*-------------------------------------------------------------*/
PRIVATE buffer *load_block( hdr *hp, unsigned req_block_abs )
{
    /* Load a block. req_block_abs holds the absolute block number  */
    /* of the requested block.                                      */

    buffer *b;
    vmem   *v = hp->next.v;

    if(b = has_buffer(v,req_block_abs)) /* if the object already    */
        touch_buffer( b, 1 );           /* owns a loaded buffer for */
    else                                /* the block, return it.    */
    {
        b       = get_buffer( v );      /* Get a new buffer.        */
        b->prev = &(v->buf);            /* at the head of the       */
        if( b->next = v->buf)           /* Attach it to the object  */
            b->next->prev = &(b->next); /* linked list.             */
        v->buf = b;
        touch_buffer( b, 0 );           /* Add it to the active     */
                                        /* queue.                   */

        /* If get_buffer() gave us back another object's buffer     */
        /* and the buffer is dirty, flush it before reading.        */

        if( b->dirty &&
            !block_io(WRITE, b->cblock, b->b.buf))
                return NULL;

        if( !block_io(READ, req_block_abs, b->b.buf))
            return NULL;

        b->dirty  = 0;
        b->cblock = req_block_abs;
    }
    return b;
}
```


PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

buffer manager at any time. You must now always use **vwrite** to modify an array cell. The modified versions of the other functions are in Listing 5. I've added one function, **has_buffer** (at the top of Listing 5), which checks to see if an object already owns a buffer for the required block.

To save space, I've also removed most of the comments from the original routines, so it might be helpful if you have the original routines in front of you as you read Listing 1. The comments that are in Listing 1 describe the changes that I've made.

A FEW CAVEATS

There are a few caveats with the new set of routines. First, the pointers returned from **vread** and **vwrite** are now valid only until the next **vread** or **vwrite** call, even if that call accesses another object entirely. This is because the buffer manager can steal a buffer from any object at any time. Second, the **vdirty** call has been removed from the package because there's no easy way to mark the buffer. You could add a **vdirty** that was passed both an object pointer and an index, which would search for the correct buffer in the linked list of buffers and then mark it, but that's an awful lot of work with little benefit.

Another potential glitch in the current system is the queueing strategy that is utilized to find the least-recently-used buffer. The remove/enqueue operation is relatively expensive—more so than a simple time-stamp would be. (It's not necessary to stamp a buffer with the real *time*; remember that a **long int** that's incremented every time that it's used to stamp an object would perform the same way.) The expense is minimized because a buffer that's already at the tail of the queue is not requeued. This means that a cluster of reads or writes to cells in the same buffer will not cause a requeueing.

If your reads or writes are scattered around the array, though, you'll need to requeue on every read or write. In this situation, it would probably be better to use a time-stamp strategy and search down the linked list of active buffers looking for the one with the oldest (smallest) time-stamp value. This traversal, of course, makes the process of stealing a buffer much less efficient.

New Version 3.0

Commenting Disassembler!

SOURCER™ 486

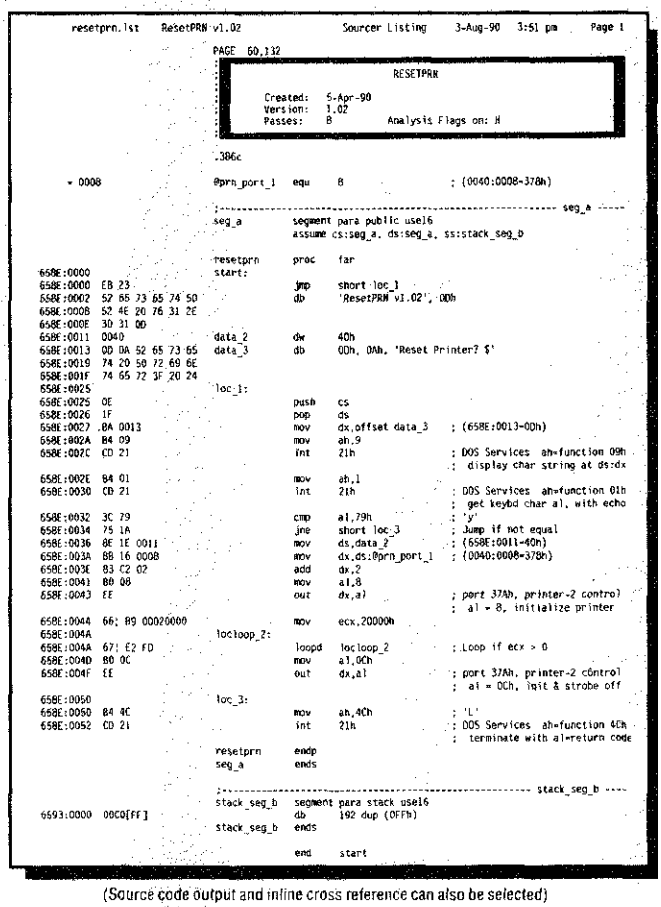
- **SEE HOW PROGRAMS WORK**
- **EASILY MODIFY PROGRAMS**

SOURCER™ creates detailed commented source code and listings from memory and executable files. Built in data analyzer and simulator resolves data across multiple segments and provides detailed comments on interrupts and subfunctions, I/O ports and much more. Determines necessary assembler directives for reassembly. Includes a definition file facility to include your own remarks and descriptive labels, force data types, and more. Complete support for 8088/87 through 80386/387, 80486, and V20/V30 instruction sets. We welcome comparisons with any other product, because no product comes close to the ease of use and output clarity of SOURCER.

Sourcer is the best disassembler we've ever seen!
—PC Magazine, January 17, 1989, page 101

SAMPLE OUTPUT

Fully automatic
Program header
Assembler directives
Determines data areas and type
Detailed comments
Simulator follows segment changes
80386 and 80486 support
Easy to read format



```
resetprn.lst    ResetPRN v1.02              Sourcer Listing    3-Aug-90  3:51 pm    Page 1

                                PAGE  60,132
                                        RESETPRN

                                Created:   5-Apr-90
                                Version:   1.02
                                Passes:    8          Analysis Flags on: H

                                .386c

= 0008                          @prn_port_1   equ     8                   ; (0040:0008-378h)

                                ;--------------------------------------------------------- seg_a -----
                                seg_a         segment para public use16
                                              assume cs:seg_a, ds:seg_a, ss:stack_seg_b

                                resetprn      proc    far
                                start:
658E:0000                       ;
658E:0000  EB 23                              jmp     short loc_1
658E:0002  52 65 73 65 74 50                  db      'ResetPRN v1.02', 0Dh
658E:0008  52 4E 20 76 31 2E
658E:0010  30 32 0D
658E:0011  0040                 data_2        dw      40h
658E:0013  0D 0A 52 65 73 65    data_3        db      0Dh, 0Ah, 'Reset Printer? $'
658E:0019  74 20 50 72 69 6E
658E:001F  74 65 72 3F 20 24
658E:0025                       ;
658E:0025  0E                 loc_1:
658E:0026  1F                               push    cs
658E:0027  BA 0013                          pop     ds
658E:002A  B4 09                            mov     dx,offset data_3    ; (658E:0013-0Dh)
658E:002C  CD 21                            mov     ah,9
                                            int     21h                 ; DOS Services  ah=function 09h
                                                                        ;  display char string at ds:dx
658E:002E  B4 01                            mov     ah,1
658E:0030  CD 21                            int     21h                 ; DOS Services  ah=function 01h
                                                                        ;  get keybd char al, with echo
658E:0032  3C 79                            cmp     al,79h              ; 'y'
658E:0034  75 1A                            jne     short loc_3         ; Jump if not equal
658E:0036  8E 1E 0011                       mov     ds,data_2           ; (658E:0011-40h)
658E:003A  8B 16 0008                       mov     dx,ds:@prn_port_1   ; (0040:0008-378h)
658E:003E  83 C2 02                         add     dx,2
658E:0041  B0 08                            mov     al,8
658E:0043  EE                               out     dx,al               ; port 37Ah, printer-2 control
                                                                        ;  al = 8, initialize printer
658E:0044  66: B9 00020000                  mov     ecx,20000h
658E:004A                       locloop_2:
658E:004A  67: E2 FD                        loopd   locloop_2           ; Loop if ecx > 0
658E:004D  B0 0C                            mov     al,0Ch
658E:004F  EE                               out     dx,al               ; port 37Ah, printer-2 control
                                                                        ;  al = 0Ch, init & strobe off
658E:0050                       loc_3:
658E:0050  B4 4C                            mov     ah,4Ch              ; 'L'
658E:0052  CD 21                            int     21h                 ; DOS Services  ah=function 4Ch
                                                                        ;  terminate with al=return code
                                resetprn      endp
                                seg_a         ends

                                ;--------------------------------------------------------- stack_seg_b ----
                                stack_seg_b   segment para stack use16
6593:0000  00C0[FF]                         db      192 dup (0FFh)
                                stack_seg_b   ends

                                              end     start
```

(Source code output and inline cross reference can also be selected)

BIOS SOURCE

- **CHANGE AND ADD FEATURES**
- **CLARIFY INTERFACES**

for PS/2, AT, XT, PC, and Clones

The BIOS Pre-Processor™ with SOURCER provides the first means to obtain accurate legal source listings for any BIOS! Identifies entry points with full explanations. Resolves PS/2's multiple jumps for improved clarity. Provides highly descriptive labels such as "video_mode" and much more. Fully automatic.

SOURCER Commenting disassembler	$129.95	**ASMtool 486**™ Assembly source analyzer and flowcharter	$199.95
SOURCER with BIOS Pre-Processor	169.95	**View-It**™ High speed zoomable file viewer	69.95
UNPACKER™ Unpack packed EXE files	39.95	**ASM ProPak II**™ Sourcer w/BIOS, UnPacker, ASMtool and View-It	399.75

Shipping & Handling: USA $5; Canada/Mexico $10; Other $15; CA Res. add sales tax; PS/2 trademark of IBM Corp.

NOT COPY PROTECTED **30-DAY MONEY-BACK GUARANTEE**

VISA **1-800-648-8266** MasterCard

V COMMUNICATIONS, INC.

4320 Stevens Creek Blvd., Suite 275, Dept. PJ8, San Jose, CA 95129 (408) 296-4224

Circle **125** on Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Writing an ANSI C preprocessor

Implementation problems in complying with the new ANSI C standards

By W. Sumerlin

In the old days, a reasonable C preprocessor read an input file, discarded comments and excess white space (spaces, tabs, and new lines), built tokens (character sequences), substituted definitions, and expanded macro definitions. It didn't care when expansions and substitutions were done. Probably the most challenging aspect of the preprocessor was handling macro expansions and argument substitutions in a constrained memory space. The PC presented two problems: segmented memory architecture due to the Intel microprocessor's 64K-byte maximum segment size and DOS's 640K maximum memory size.

Now, with the new ANSI C standard, there are new operators, new preprocessing commands, and new specific size limits (some quite large). As a result, the preprocessor has to be sensitive to when it can and cannot expand macro definitions. Due to the new requirements, small design and implementation changes in the preprocessor can have dramatic effects on whether the preprocessor meets or misses the ANSI C standard.

A great deal has been written about what comprises the new ANSI standard, but you won't see much about the ease or difficulty of implementing that standard in C compilers or supporting tools. While some steps toward making C programming simpler to use and understand are easy to implement, others are real challenges. Seemingly minor changes and additions have made current designs and implementations totally obsolete.

The new ANSI C standard adds several preprocessor commands, supplies two new macro operators, and clarifies how macro expansion is done. The standard's goal is to make every manufacturer's implementation of a C preprocessor behave in the same fashion. This does *not* mean similar, it means *identical*. If the preprocessors were not the same, small differences could cause different code to be generated (unbeknownst to you) or even create fatal compilation errors. If you didn't have access to what the preprocessor was generating, you might not be able to identify—much less fix—the compilation problem.

The ANSI standard is described in two documents. The first provides rules and examples, while the second describes why some things were done the way they were. If you happen to appreciate dry, technical reading material, you'll enjoy the first document.

There are three main problems with the rules and descriptions of the ANSI standard. First, the rules fail to fully describe the results of preprocessing the examples that are included in the document. Second, while the suggested implementation method may work in a virtual memory space (i.e., no memory limits), we couldn't find any way it would work in a segmented memory architecture where a segment was restricted to 64K in size. And third and finally, the entire presentation appeared to be written for large system environments with no memory constraints whatsoever. This would be fine, except that there are millions of constrained memory systems in this world, and only thousands of unconstrained environments.

If you are successful in implementing the ANSI standard, the following code fragment (as taken from the standard's document):

```
#define x    3
#define f(x) f(x * (a))
#undef x
#define x    2
#define g    f
#define z    z[0]
#define h    g(~
#define m(a)         a(w)
#define w    0,1
#define t(a) a

f(y+1) + f(f(z)) % t(t(g)(0) + t)(1);
g(x+(3,4)-2) | h 5) & m
        (f)^m(m);
```

Bill Sumerlin is commonly found poring over piles of C code at MMC AD Systems, where they specialize in tools for C programmers on the PC, Macintosh, and Sun workstations. His address is MMC AD Systems, Box 360845, Milpitas, CA 95036.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

For Project Managers
Developing In C on a LAN

Developing software on a LAN is fast, efficient, and dangerous.

Sage makes it safe, controllable and auditable.

Experienced managers are rightly concerned about the integrity of LAN development environments. It's too easy to overwrite or corrupt files.

To safely develop on a LAN you need control over file changes, the ability to recover any module revision or system version, audit trails, and security that doesn't hamper productivity.

The solution: the Sage LAN development environment, three tools that operate independently or snap together synergistically.

The primary element is Sage PVCS which provides version control and change management features, file and function security, automatic audit trail and detailed project reports.

The best way to access PVCS is via the Sage Professional Editor (SPE), an extensible control center that simplifies access to any tool and provides industrial-strength editing. SPE invokes PVCS automatically. Just open modules to edit and the PVCS window pops open.

The final element in the Sage LAN development environment is PolyMake - a sophisticated tool for automatically performing correct builds. A single command recompiles all modules that have been changed, re-links as necessary, and produces a correct, conforming program.

PVCS: $495- MS-DOS, Single User.
$495- OS/2, Single User.
PolyMake: $179- MS-DOS, Single User.
$179- OS/2, Single User.
Sage Professional Editor: MS-DOS, OS/2 and Dual Mode included in one package. $295- Single User.
Prices are U.S. list. Call for Dual Mode, network, site license and package prices.
Visa, MC & P.O. orders:

1-800-547-4000

30 Day Money Back Guarantee

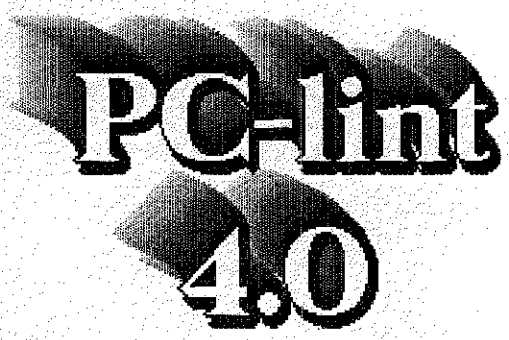
The Leader In
Desktop Development Tools

SAGE SOFTWARE

Sage Software, Inc., 1700 NW 167th Place, Beaverton, OR 97006 (503) 645-1150, Fax (503) 645-4576, Int'l (301) 230-3307, Int'l Fax (301) 984-3047.

Circle 26 on Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED



The Best Lint Just Got Better

PC-lint is a C source code analysis tool. It will check code more thoroughly than a C compiler and, because it looks across multiple modules, it enjoys a perspective that your compiler doesn't have.

Control your development time and cost. Find C source code errors before they find you.

Over **270 error messages**. More than **90 options** for complete customization. Suppress error messages, locally or globally, by symbol name, by message number, by filename, etc. **Easy to Use** - If you know C you already know how to use PC-lint. Check for **portability** problems. Alter size of scalars. Adjust format of error messages. Automatically generate ANSI **prototypes** for your K&R functions.

New Features in Version 4.0

- **Lint Object Modules** - incremental linting - big linting.
- **Weak Definial Checking** - macros, structs, enums, typedef's and declarations not used globally and locally. Header files not used. Externals that can be made static. Declarations that can be moved from headers. etc.
- **Library Headers** - avoid redundant prototype processing.
- **Over 70 new messages and 30 new options,** including identifier truncation clashes.
- **New "Elective Notes" message category.**
- **Unix lint compatibility**

Mainframe and Mini Programmers

Get **FlexeLint 4.0** (formerly Generic Lint) in shrouded source form for VAX/VMS, Unix, IBM VM/MVS, QNX OS-9, DG AOS/VS, etc. Requires only K&R C to compile but supports ANSI. Pricing starts at $798. Call for details.

Gimpel Software

3207 Hogarth Lane Collegeville PA 19426

(215) 584-4261 - CALL TODAY

(215)584-4266 - FAX

PC-lint is only $139 - Specify MS-DOS or OS/2

30 Day Money-back Guarantee

Quantity and educational discounts available. PA add 6% sales tax. Outside USA add $20.

PC-lint, FlexeLint and Generic Lint are trademarks of Gimpel Software.

Contact this advertiser directly.

when preprocessed, becomes:

```
f(2 * (y+1)) + f(2 * (f(2 * (z[0])))) % f(2 * (0)) + t(1);
f(2 * (2+(3,4)-0,1)) | f(2 * (~5)) & f(2 * (0,1))^m(0,1);
```

Once the implementation has successfully generated the previous result, the following code fragment:

```
#define str(s)          # s
#define xstr(s)     str(s)
#define glue(a, b)  a ## b
#define xglue(a, b) glue(a, b)
#define HIGHLOW         "hello"
#define LOW             LOW ", world"
glue(HIGH, LOW)
xglue(HIGH, LOW)
```

is supposed to be translated into:

```
"hello"
"hello" ", world"
```

But, the first time you usually get:

```
"hello"
"hello"
```

When you make the necessary adjustments to get the proper result, the original case has been broken. To make matters worse, there are other pathological cases which need to be processed with a specific result.

So what is the difficulty? Macro expansion and argument substitution processing have changed as a result of four new rule refinements:

1. A macro definition without arguments is passed through without changes;

2. If a macro is currently being expanded, it cannot be expanded again (i.e., using recursion), but is instead passed through unexpanded;

3. The new ## operator affects when macro arguments are substituted and expanded. If the ## operator is present, the two surrounding symbols are combined prior to expansion—otherwise, they are expanded when encountered; and

4. Macro expansion is now context-sensitive. For example, if a macro name spans a definition and the original source code (i.e., the **t(1)** result in the first example above), no expansion or substitution occurs.

SOME RULE APPLICATIONS

Let's apply the previous rules to some source code examples. By applying these rules to real cases, we can begin to develop an algorithm of how to process the code.

The following example demonstrates a seemingly simple macro definition argument substitution:

```
#define x   2
#define f(a) f(x * (a))

f(y + 1);
```

Using the **x** and **f(a)** macro definitions, macro expansion of **f(y +1)** yields **f(2 * (y + 1))**, as follows:

```
1:    f(y + 1)
2:          --> f(x * (y + 1))       /* using f(a) definition   */
3:              --> f(2 * (y + 1))   /* using x definition      */
```

Why don't we continue expanding the **f** macro in Step 3? We are already expanding the **f** macro definition from Step 1, so we do *not* expand the resultant **f** macro definition because, as

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Rule 2 states: *If a macro is currently being expanded, it cannot be expanded again.*

So far, so good. Let's introduce another definition to the previous set of definitions:

```
#define z    z[0]
```

and expand **f(f(z))**:

```
1:      f(f(z))
2:          --> f(2 * (f(z)))                 /* using f(a) on the 1st f() */
3:              --> f(2 * (f(2 * (z))))       /* using f(a) on the 2nd f() */
4:                  -->f(2 * (f(2 * (z[0])))) /* using z definition        */
```

This seems to be a pretty straightforward expansion of the macro definition from the outside to the inside. But what happens if we use an alternate approach and expand the macro definition from the inside out?

```
1:      f(f(z))
2:          --> f(f(z[0]))                    /* using z definition          */
3:              --> f(f(2 * (z[0][0])))       /* using f() and z definitions */
4:                  -->f(2 * (f(2 * (z[0][0][0])))) /* using f() and z       */
```

The result is different from that obtained in the previous approach. Why did we continue to expand the **z** definition in each step while we did *not* expand the **f** macro definition? We didn't expand the embedded **f**'s each time because we were in the process of expanding the outer **f** macro definition and recursion is not allowed, per Rule 2. But, though we already expanded the **z** definition, none of the rules prevent us from expanding the **z** definition again. Even if we didn't *want* to expand the **z** definition again, since we previously expanded it as an embedded argument, we could not have prevented it without using an excessive amount of storage to remember the context of each character during expansion. When working in a constrained environment where the maximum token size is 32K (ANSI standard) and only 640K memory is available (PC DOS), the requisite storage is not available. Hence, *the "inside out" expansion approach will not work with this code fragment.*

Now let's add two more definitions:

```
#define g       f
#define t(a)    a
```

and continue expanding the next fragment in the example from the standard:

```
1:      t(t(g)(0) + t)(1)
2:          --> t(g)(0) + t(1)                /* using t() on the 1st */
3:              -->g(0) + t(1)                /* using t() on the 2nd */
4:                  --> f(0) + t(1)           /* using g definition   */
5:                      --> f(2 * (0))+ t(1)  /* using f()            */
```

Why didn't we expand the **t(1)** macro at the end of the line starting at Step 3? Following Rule 4, we did not expand a macro that was not complete within a macro definition. The difficulty in this particular case is figuring out how we are supposed to know that the **t** macro is incomplete. If we simply push the macro definition back onto the input, as many previous preprocessor implementations did, there would be no way to distinguish between what comes from the input file and what is part of the macro definition and arguments. The new standard seems to imply that we should use a "mainframe" approach in which we have an infinite amount of memory (either real or virtual). Using this approach, we would need to

ANSI C Standard Update

At long last, the American National Standard for C has been approved. The Standard's official designation is ANSI X3.159-1989. To obtain a copy, contact:

American National Standards Institute
Sales Department
1430 Broadway
New York, NY 10018
Tel.: (212) 642-4900; Fax: (212) 302-1286

The X3J11 committee is currently in its interpretation phase and is obliged to respond to all formal requests for interpretation of the Standard. If you would like to submit a request for interpretation, write to:

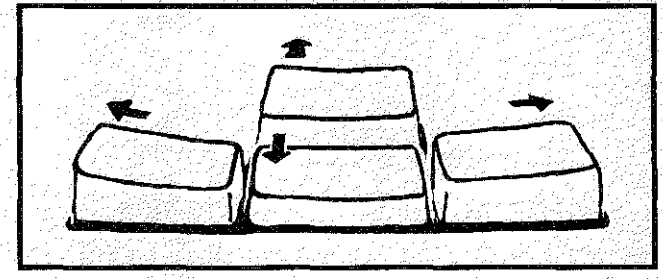
X3 Secretariat CBEMA
311 First Street N.W., Suite 500
Washington, D.C. 20001-2178
Attn: Manager of Standards Processing

I attended an ISO C meeting in London in June as the international representative for the U.S. There is every chance that the ISO C standard will be finalized this year and that it will be identical to ANSI's. However, over the next few years several addenda will likely become part of the ISO standard. The main addendum is expected to deal with multi-byte character extended support in the library.

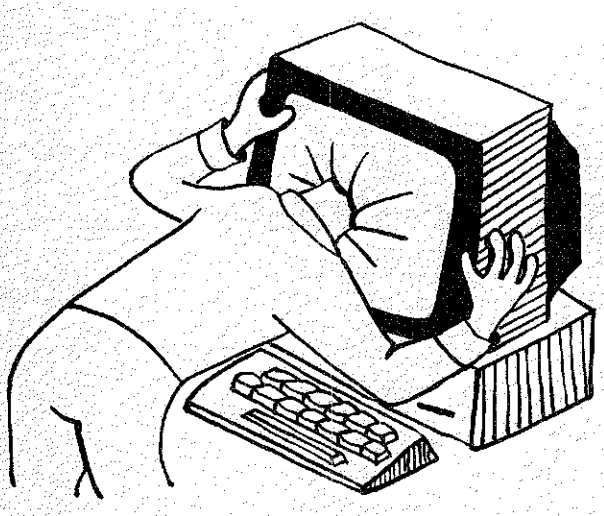
There is no approved validation suite for the U.S. government yet, so vendors who claim to be ANSI-conformant are primarily asserting that they pass one or more of the commercial suites available. In most cases these claims are either not completely true (read "marketing claims") or the suite they're using is incomplete. My own mini-suite has found shortcomings in *every* DOS C compiler tested to date. I do believe, however, that the mainstream compiler vendors really are trying to be ANSI-conformant, and that the areas in which they fall short are typically few and more the results of oversights than of glaring omissions.

The Numerical C Extensions Group (NCEG), which I started in March of 1989, is alive and well and has tentatively been accepted as a working group within X3J11. We continue to meet every six months. The most active subgroups are: complex arithmetic, variable dimensioned arrays, IEEE support, aliasing, and vector/array operations. I am delighted that Dennis Ritchie, the principal designer of C, has become active in NCEG. He recently proposed an alternate design for variable dimensioned arrays.

—By Rex Jaeschke



PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED



Look inside your software!

Quaid Analyzer shows the inner workings of software, even without the source. Scroll through text or instructions as easily as with an editor.

You can get a report on the screen at every DOS service call, or every interrupt or breakpoint.

Ordinarily **Quaid Analyzer** is memory resident. For demanding tasks, load it before DOS, then watch DOS load and install device drivers. **Quaid Analyzer** includes ways of tracing in spite of countermeasures used to make tracing difficult. We use it to unravel copy protection. **Quaid Analyzer** is pure software, so you don't have to take apart your computer.

To order **Quaid Analyzer**, call us with your credit card, or send us a check for $200 US funds. We ship within a day at our expense.

Quaid Analyzer comes with a manual, and software on a 3 inch and a 5 inch diskette. It is not copy protected. If you are not satisfied, you can return it within 30 days for a refund.

Quaid Software Limited

45 Charles Street East, Third Floor Dept P
Toronto Ontario Canada M4Y 1S2

(416) 961-8243 fax: (416) 961-6448

Contact this advertiser directly.

segregate the macro-text processing and expansion from the normal input-text processing and perform all of our work on the macro and arguments. When done, we would pass the resultant text stream on prior to continuing to process the input file. This approach implies that we:

- Maintain two separate but similar sets of routines (e.g., **get_char**, **push_char**, ...) that work from different data sources [difficult to support, additional memory]; or
- Maintain one set of routines, but have them switch back and forth between two different input sources [additional memory, additional temporary storage on small systems].

The important thing to note is that the standard seems to state that you must completely expand a macro prior to continuing on to process the input.

THE FINAL RESULT

It took us three attempts to get our C preprocessor, Cpp, to function properly in all the ANSI cases. It wasn't easy. We aren't even quite sure how or why it works. All we know is that it does work.

All classical design/implementation approaches failed. We ended up using a combination of approaches that included input-text push-back and keeping track of the origins of the characters. Our implementation runs in small, segmented architecture (IBM PC) as well as in Sun's virtual memory environment. Because of the complexity of the implementation, the preprocessor doesn't run as fast as we would like, but it's still fast. In addition to providing ANSI support, it provides support for a number of the non-ANSI compiler implementations, so Cpp supports both old and new code.

As with all new code, our preprocessor has had its share of bugs. Initially, the biggest problem was that we found the eight conditional levels allowed for by the ANSI standard far too restrictive. Every C compiler's system files on the Macintosh exceeded the eight levels. We even found one compiler's "include" files which contained over 30 levels! We also ran into a problem with trigraphs, since "????" is a common character sequence on the Macintosh.

We had our implementation bugs as well. For instance:

```
#define x p->x x;
```

would cause the preprocessor to hang in an infinite loop while it attempted to continuously expand **x**. In some cases we compressed too much space out of the macro definition. You need to hold onto or even insert a space at the end of a macro definition if it will be embedded later in another unique macro definition. If you don't, two symbols can be appended together and form a new, undesired symbol. We also messed up floating point numbers in a couple of very obscure forms.

Finally, we inadvertently threw away the last line of a file when it was not terminated with a new line. This may not seem like a big deal, since many editors automatically insert a new line on the last line, but we found many system "include" files of one particular compiler manufacturer that had unterminated last lines. We got a lot of calls, to put it mildly.

Given our experience with the C preprocessor, we salute all those with working, fully compliant ANSI C preprocessors. We only have one question: how many of you use the new preprocessor features, and for what purpose? PJ

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Tear Your C Development Time in Half.

We understand the pressure you're under, trying to create, test, and debug complex C programs without Instant-C™, the professional development environment. With all your searching for errors and recompiling, you barely have time to test code, never mind read magazines.

Instant-C can make your time twice as productive. Here's how:

Industrial-strength debugging. Instant-C finds bugs other compilers miss, automatically detecting over 500 errors, including:

- *Array references out of bounds*
- *Stack overflow*
- *Use of uninitialized pointers*
- *Overwriting a return address*
- *Read or write through null pointers*
- *Overwriting your program or DOS*

Ultra-fast, incremental compiling. When you make a change to a program, Instant-C recompiles just the change. In fact, Instant-C can rebuild a 100,000 line program in less time than it takes to read our headline. And in recent tests, Instant-C was 24 times faster than Microsoft Programmer's WorkBench.*

Microsoft® C compatibility. Instant-C links with Microsoft C libraries to form an enhanced development environment. After developing, testing, and debugging with Instant-C, you can use Microsoft C to make the final version of your program.

Unlimited program size. Instant-C uses Rational Systems' DOS/16M technology, the software that enabled Lotus to break the 640K barrier for 1-2-3® Release 3. Because it uses extended memory, no program is too big, and your work isn't slowed down by swapping to disk.

That's not the half of it. Instant-C is a complete development environment, including source-level debugging, dynamic cross referencing, lint-like static check, support for third-party libraries, and interactive testing of single functions or partial programs.

The Instant-C Environment

- The industry's fastest edit-compile-link-debug cycle
- Configurable editor and source formatter
- Incremental compiler and linker
- Direct access to 16MB of memory
- Extensive static and run-time error checking
- Multiple virtual or real screens
- Source-level debugging
- Support of commercial libraries
- Conditional breakpoints in C
- Compatibility with assembly language object modules
- Immediate evaluation of C expressions, including macros
- Compatibility with Microsoft C compiler 5.1 and 6.0
- Interactive execution of functions
- Allows use of Microsoft run-time libraries
- Dynamic cross-referencing and browsing
- Full graphics screen support

Get the whole story. To find out more about how Instant-C can double your productivity, give us a call. We'll give you all the details in a flash. After all, we know how valuable your time is.

Rational Systems, Inc.

Rational Systems, Inc.
220 N. Main St.
Natick, MA 01760
Toll-free: (800) 666-2585
Outside U.S.: (508) 653-6006
Fax: (508) 655-2753

* Tests involved changing one line of code in a 11,500-line program, running on a 20MHz 80386 system with 8MB memory. Instant-C requires an 80286/386/486 and at least 1M of memory. Instant-C is a trademark of Rational Systems, Inc. Microsoft is a registered trademark of Microsoft Corp. 1-2-3 is a registered trademark of Lotus Development Corp.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

A look at three major 80486 compiler optimizations

Peak performance: on to the 80486

By Mark J. Barrenechea

What does it mean to have a compiler optimized for the 80486? Why should a developer spend the extra money for an 80486 compiler when an 80386 will do? What performance benefits will be gained by moving an application to the 80486? This article addresses these questions, primarily with the help of Intel's *i486 Microprocessor Programmer's Reference Manual*.

The Intel 80486 is 100% binary-code compatible with the Intel 80386. Any 8086/8088, 80286, or 80386 application will execute on an 80486 machine without modification, recompiling, relinking, or rebinding.

Very few differences exist between the 80386 DX or SX and the 80486 processors, as far as the application developer is concerned. The 80486 processor defines:

- a new 8K-byte unified code and data cache
- new instruction speeds
- an internal 80487 floating point unit
- a new flag (AC)
- a new exception (vector 17)
- 3 new application-level instructions
- 6 new control register bits
- 2 new page-table entry bits, and
- a change in segment descriptor loads.

The 80486 has obviously been engineered for higher performance than the 80386. Many of the 80386 instructions execute faster on the 80486, often in one clock cycle. However, there are code optimizations designed to make programs execute faster on an 80486 which yield only minor performance penalties when they are run on an 80386. These optimizations include alternate instruction sequence selection for the 80486, instruction scheduling, a new procedure

Mark J. Barrenechea is director of language development at MicroWay, Inc. in Plymouth, Massachusetts.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

LISTING 1

```
void foo(int i, int *ptr){
   ...
   printf(" %d %d %x \n",3,i,ptr);
   ...
   }
```

LISTING 2

```
; Old-Style Calling Convention

        push    ebp
        mov     ebp,esp
        sub     esp,8
        ...
        mov     eax,3                   ; 1 clock cycle
        push    dword ptr [ebp]+12      ; 3 clock cycles
        push    dword ptr [ebp]+8       ; 3 clock cycles
        push    eax                     ; 3 clock cycles
        lea     edx,dword ptr ds:L5     ; 1 clock cycle
        push    edx                     ; 3 clock cycles
        call    _printf                 ; 3 clock cycles
        add     esp,16                  ; 3 clock cycles
        ...
        mov     esp,ebp
        pop     ebp
        ret

; New-Style Calling Convention

        push    ebp
        mov     ebp,esp
        sub     esp,32
        ...
        lea     eax,dword ptr [ebp]+12  ; 1 clock cycle
        mov     dword ptr [ebp]-16,eax  ; 1 clock cycle
        lea     eax,dword ptr [ebp]+8   ; 1 clock cycle
        mov     dword ptr [ebp]-20,eax  ; 1 clock cycle
        mov     dword ptr [ebp]-24,3    ; 1 clock cycle
        lea     eax,dword ptr ds:L5     ; 1 clock cycle
        mov     dword ptr [ebp]-28,eax  ; 1 clock cycle
        call    _printf                 ; 3 clock cycles
        ...
        mov     esp,ebp
        pop     ebp
        ret
```

LISTING 3

```
/* Consider the following simple C program that mixes
   floating and integer operations: */

        foo(){
           extern float f1,f2,f3;       /* these are externs so they */
           extern int i1,i2,i3;         /* will not get optimized away! */
           f1 = 2.0;
           f2 = 4.0;
           f3 = 6.0;
           f1 = f1 * f2 + f3;
           i1 = 2;
           i2 = 4;
           i3 = 6;
           i1 = i1 * i2 + i3;
           }
```

LISTING 4

```
; Consider the following unscheduled i486 assembly code:

; NOTE: All instructions are 1 clock cycle unless noted otherwise

;   f1 = 2.0;
        mov     eax,offset [of literal pool]
        mov     edx,[eax]
        mov     f1,edx

;   f2 = 4.0;
        mov     ecx,4[eax]
        mov     f2,ecx

;   f3 = 6.0;
        mov     eax,8[eax]
        mov     f3,eax
;
; At this point, f1, f2, and f3 are loaded with the proper values
;
;   f1 = f1 * f2 + f3;
        fld     dword ptr f2        ; 3 clock cycles
        fmul    dword ptr f1        ; 11 clock cycles
        fadd    dword ptr f3        ; 10 clock cycles
        fstp    dword ptr f1        ; 7 clock cycles

;   i1 = 2;
        mov     eax,2
        mov     i1,eax

;   i2 = 4;
        mov     edx,4
        mov     i2,edx

;   i3 = 6;
        mov     ecx,6
        mov     i3,ecx
;
; At this point, i1, i2, and i3 are loaded with the proper values
;
;   i1 = i1 * i2 + i3;
        add     edx,edx             ; 3 cycles
        add     ecx,edx             ; 3 cycles
        mov     i1,ecx
```

LISTING 5

```
;   f1 = 2.0;
        mov     eax,offset [of literal pool]

;   f2 = 4.0;
        mov     ecx,4[eax]
        mov     f2,ecx

;   f1 = f1 * f2 + f3;
        fld     dword ptr f2        ; 3 clock cycles
        mov     edx,[eax]
        mov     f1,edx
        fmul    dword ptr f1        ; 11 clock cycles

;   f3 = 6.0;
        mov     eax,8[eax]
        mov     f3,eax

;   i1 = 2;
        mov     eax,2
        mov     i1,eax

;   i2 = 4;
        mov     edx,4
        mov     i2,edx

;   i3 = 6;
        mov     ecx,6
        mov     i3,ecx
        mov     eax,eax             ; nops are not necessary, they just
        mov     eax,eax             ; complete the integer core unit cycles
        fadd    dword ptr f3        ; 10 clock cycles

;   i1 = i1 * i2 + i3;
        add     edx,edx             ; 3 cycles
        add     ecx,edx             ; 3 cycles
        mov     i1,ecx
        mov     eax,eax             ; integer nop optional
        mov     eax,eax             ; integer nop optional
        fstp    dword ptr f1        ; 7 clock cycles
                                    ; 6 integer nops could go here also
```

PJ CODE

All code in this issue is available through our bulletin board services or directly from *PJ* on disk. For more information see pages 16 and 80.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

AMX

REAL-TIME MULTITASKING KERNEL

8086/88, 80x86/88 80386
Z80, 64180, 8080/85 68000/10/20

- Fast, reliable operation
- Compact and ROMable
- PC peripheral support
- DOS file access
- C language support
- Preemptive scheduler
- Time slicing available
- Configuration Builder
- Complete documentation
- Intertask messages
- Message exchanges
- Dynamic operations
  - — task create/delete
  - — task priorities
  - — memory allocation
- Event Manager
- Semaphore Manager
- List Manager
- InSight™ Debugging Tool

THE BEST

Join over 600 developers such as IBM®, Xerox, Hewlett Packard, Hayes, Hughes Aircraft and NASA.

CHOOSE AMX

The best low-cost, high-performance real-time multitasking system available today.

No Royalties
Source Code Included

Demo Disk $25 US
Manual only $75 US
AMX 86 $3000 US
(Shipping/handling extra)

Call for prices for other processors.

IBM is a registered trademark of IBM Corp.
Z80 is a trademark of Zilog, Inc.
AMX, AMX 86, InSight are trademarks of KADAK Products Ltd.

KADAK Products Ltd.
206-1847 West Broadway
Vancouver, B.C., Canada
V6J 1Y5
Telephone: (604) 734-2796
Fax: (604) 734-8114

Circle 211 on Reader Service Card

calling convention, condition codes, code and cache alignment, new instruction opportunities, and some other miscellaneous 486 issues.

ALTERNATIVE INSTRUCTIONS

There are four opportunities for optimizations with alternative instruction sequence selection on the 486. They involve constants, the **leave** instruction, string instructions, and some integer instructions. Each of these will be discussed below.

Let's look at a typical code sequence that uses a constant as the source operand in a **mov** instruction:

```
mov     x,1234h             ;both immediate and
mov     y,1234h             ;memory offset
```

The corresponding C code would be:

```
x = y = 0x1234;
```

The 486 takes an additional clock cycle to execute the first instruction because an immediate data field and a memory offset field are both used. As a rule, if a constant is to be used multiple times, it is better to load it into a register and use the register multiple times. If the constant will only be used once, it is more efficient to use the immediate data. 80386 performance is not affected by this optimization.

The following code executes faster on the 486 than the code listed above, even though this one contains an additional instruction:

```
mov  eax,1234h
mov  x,eax
mov  y,eax
```

A common instruction generated by 80386 compilers, **leave**, implements the return from a routine:

Instruction		386	486
mov	esp,ebp	2	1
pop	ebp	4	1+1 (esp penalty)
leave		4	5

The corresponding C code is:

```
void foo(void) {
    return();
}
```

On the 486, the **leave** instruction is more expensive than on the 80386 by one clock cycle, but the **mov/pop** combination saves two clock cycles over the **leave**. On the 386, the **mov/pop** combination is both slower and longer. Thus, 486 compilers should save two clock cycles in generating the **mov/pop** combination, but they do so at the expense of an extra instruction. The string instruction looks like this:

Instruction		386	486
mov	eax,[esi]	4	1
add	esi,4	2	1
lodsd		5	4

The load string instruction (**lods**) is similar to the **leave** instruction. The **mov/add** combination executes two clock cycles faster on the 80486, whereas there is a one-clock penalty on the 80386. The **lods** instruction is typically located within a loop in order to perform repeat moves of elements within a string. As a matter of fact, many of the ANSI C **mem** and **str** routines can be written in assembly language using the **lods** instruction for optimal performance. On an 80486, they should be tuned to use the **mov/add** combination.

The fourth optimization opportunity comes with the integer instructions:

Instruction		386	486
mov	reg,mem	4	1
push	reg	2	1
push	mem	5	4

The 486 executes frequently-used instructions such as register ALU operations and register loads and stores all in one clock cycle. However, the memory operations now take more clock cycles than the corresponding register instructions. This was not true on the 80386. Consider the **push mem** instruction: the 486 compilers will generate a load from memory into a register before doing a **push**. This results in a two-clock savings, whereas a single clock cycle is lost on the 386. This type of code sequence is often seen for function calls. Even more clock cycles can be saved by employing a new calling convention for the 486, described below.

MORE CREATIVE CALLING

One of the more creative 80486 optimizations, **mov**, was first implemented by MetaWare in High C; other compiler vendors soon followed suit. The **mov** instruction will run in one clock cycle on the 486 when its operand is in the cache. Because of this, performance can be enhanced when passing arguments to a routine by avoiding the **push** instruction. Instead of using the standard C

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

calling convention of pushing arguments in reverse on the stack, a fixed-size "build area" is allocated within the routine's prolog that is large enough to contain the maximum argument list (see your compiler manual for complete details). This "build area" is addressed by **esp** because it is placed at the end of the stack frame.

Notice that the stack no longer needs to be popped after the call because no arguments have been pushed. Also, no additional libraries are needed because the stack still looks the same to the called routine.

The only potential extra baggage is the difficulty in debugging a program should a run-time exception occur. Because the **ebp** register is no longer needed as part of the calling convention, a compiler may map variables or constants to **ebp** just as it would to any other general-purpose register. In the Intel convention, **ebp** pointed to the base of the relevant active stack frame. In the new convention, this is no longer true. As a result, a call trace through the **ebp** register is now impossible when a run-time exception occurs.

Consider the old and the new style calling conventions for the C program described in Listing 1. Note that the old convention took 20 clock cycles, while the new convention takes only 10—a savings of 50 percent! (See Listing 2.) In languages that are procedure-intensive, such as C, C++, and Pascal, this 486 optimization is invaluable.

INSTRUCTION SCHEDULING

The floating point unit on the 486 is a separate execution unit of the processor. This was also true of the 386/387. Floating point and integer instructions can execute in parallel as a result. Therefore, any instruction sequence that will allow both units to execute in parallel will run faster.

Instruction scheduling is a non-trivial process. It requires building a direct acyclic graphic of instructions to determine a proper reordering so as to take full advantage of the parallel nature of the 486. Those instructions that have resource conflicts, i.e., register conflicts, cannot be randomly reordered.

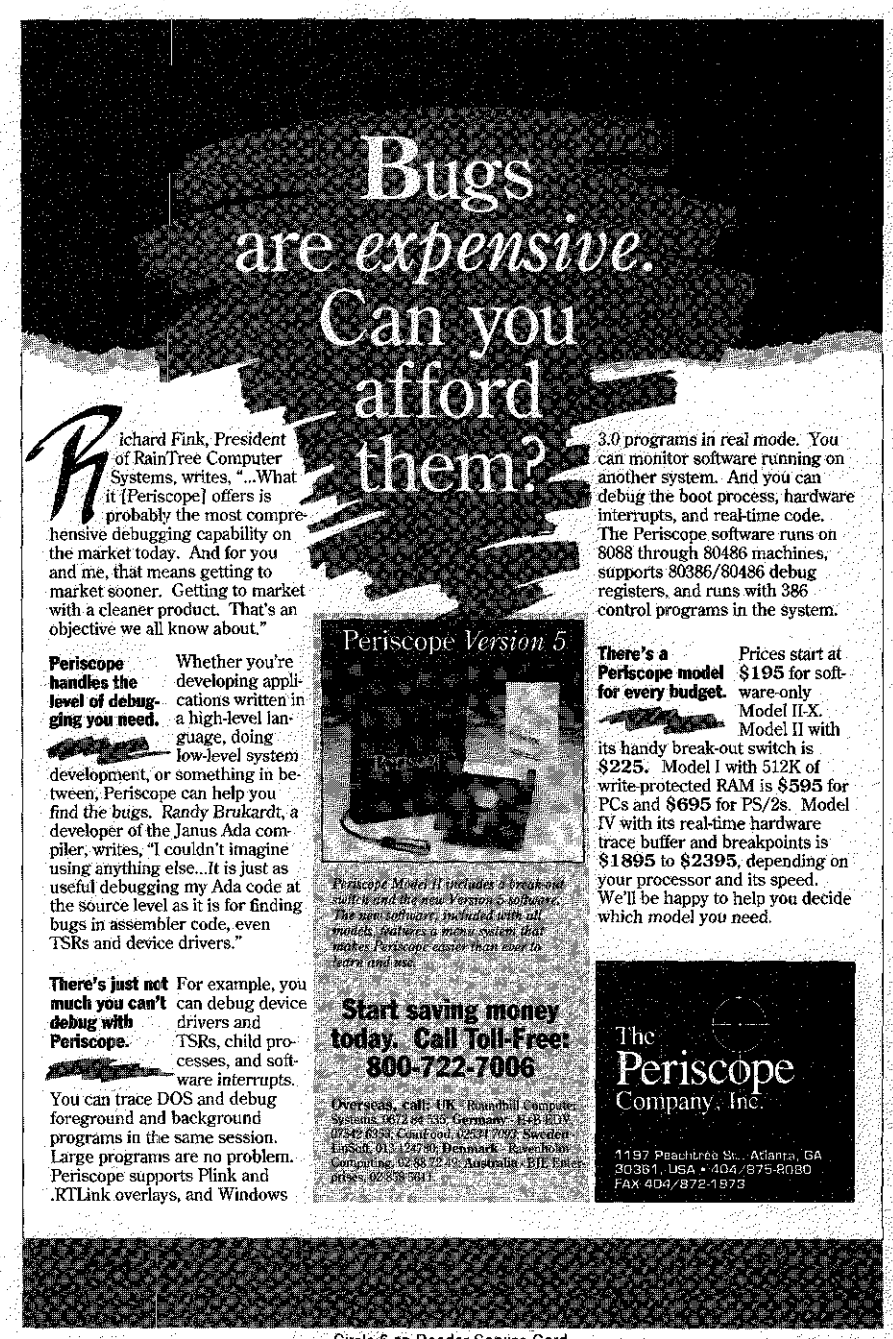
The best way to envision this is by looking at the example that is given in Listing 3. The primary opportunity for optimizing the 486 code in Listing 4 centers around the block of floating point operations:

```
fld     dword ptr f2      ; 3 clock cycles
fmul    dword ptr f1      ; 11 clock cycles
fadd    dword ptr f3      ; 10 clock cycles
fstp    dword ptr f1      ; 7 clock cycles
```

If the instruction after the **fld** were an integer instruction, the two instructions would execute concurrently. Since the next instruction is a floating point operation, the integer core unit sits idle. In fact, the entire code fragment above takes 51 clock cycles while the integer unit sits idle for 21 cycles—a loss of 41% throughout.

The scheduled instruction sequence in Listing 5 takes 34 clock cycles to execute, a net savings of 33% over the unscheduled instructions.

A disadvantage to instruction scheduling is that it is difficult to read and debug the scheduled instructions. Because high-level constructs such as adds



Richard Fink, President of RainTree Computer Systems, writes, "...What it [Periscope] offers is probably the most comprehensive debugging capability on the market today. And for you and me, that means getting to market sooner. Getting to market with a cleaner product. That's an objective we all know about."

Periscope handles the level of debugging you need. Whether you're developing applications written in a high-level language, doing low-level system development, or something in between, Periscope can help you find the bugs. Randy Brukardt, a developer of the Janus Ada compiler, writes, "I couldn't imagine using anything else...It is just as useful debugging my Ada code at the source level as it is for finding bugs in assembler code, even TSRs and device drivers."

There's just not much you can't debug with Periscope. For example, you can debug device drivers and TSRs, child processes, and software interrupts. You can trace DOS and debug foreground and background programs in the same session. Large programs are no problem. Periscope supports Plink and .RTLink overlays, and Windows 3.0 programs in real mode. You can monitor software running on another system. And you can debug the boot process, hardware interrupts, and real-time code. The Periscope software runs on 8088 through 80486 machines, supports 80386/80486 debug registers, and runs with 386 control programs in the system.

Periscope Model II includes a break-out switch and the new Version 5 software. The new software, included with all models, features a menu system that makes Periscope easier than ever to learn and use.

There's a Periscope model for every budget. Prices start at $195 for software-only Model II-X. Model II with its handy break-out switch is $225. Model I with 512K of write-protected RAM is $595 for PCs and $695 for PS/2s. Model IV with its real-time hardware trace buffer and breakpoints is $1895 to $2395, depending on your processor and its speed. We'll be happy to help you decide which model you need.

Overseas, call: UK - Roundhill Computer Systems, 0672 54 535; Germany - ELB-EDV, 07342 6338; Courtaboeuf, (2)571 7093; Sweden - LinSoft, 013-124780; Denmark - Rosenholm Computing, 02 88 72 45; Australia - BJL Enterprises, 02 858 5611.

Circle 6 on Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

A disadvantage to instruction scheduling is that it is difficult to read and debug the scheduled instructions.

and assignments can typically generate multiple assembly statements, these statements can be broken up and spread out. A typical reaction to reading scheduled code is, "How did that instruction get there?" Because of this, turning off scheduling until a procedure is fully debugged may prove to be the prudent approach to take.

CACHE AND CODE ALIGNMENT

The destination of any **jump**, **call**, or **ret** instruction should be aligned. This is because the instruction prefetch unit can be filled as soon as it is emptied. On the 80386 DX processor, these instructions should be aligned on a 0-mod-4 byte address because fetches are done four bytes at a time. On the 80486 processor, these instructions should be aligned on a 0-mod-16 byte boundary because fetches are done 16 bytes at a time due to the on-chip cache. The only trade-off is code size. By aligning on a 0-mod-16 byte boundary, code size will grow.

What is the savings? The penalty on the 486 can be up to five clock cycles to refresh the cache if the instruction is not properly aligned. If, however, the target of the instruction is already in the cache, then the penalty is two clock cycles to refresh the prefetch unit.

What can be done with this "hole" in the code? One of two things can happen. Either a **nop** instruction can be inserted, or the instructions can be reordered so as to naturally align them on 0-mod-4 or 0-mod-16 byte addresses through instruction scheduling (as discussed above). This amounts to filling the hole with a useful instruction.

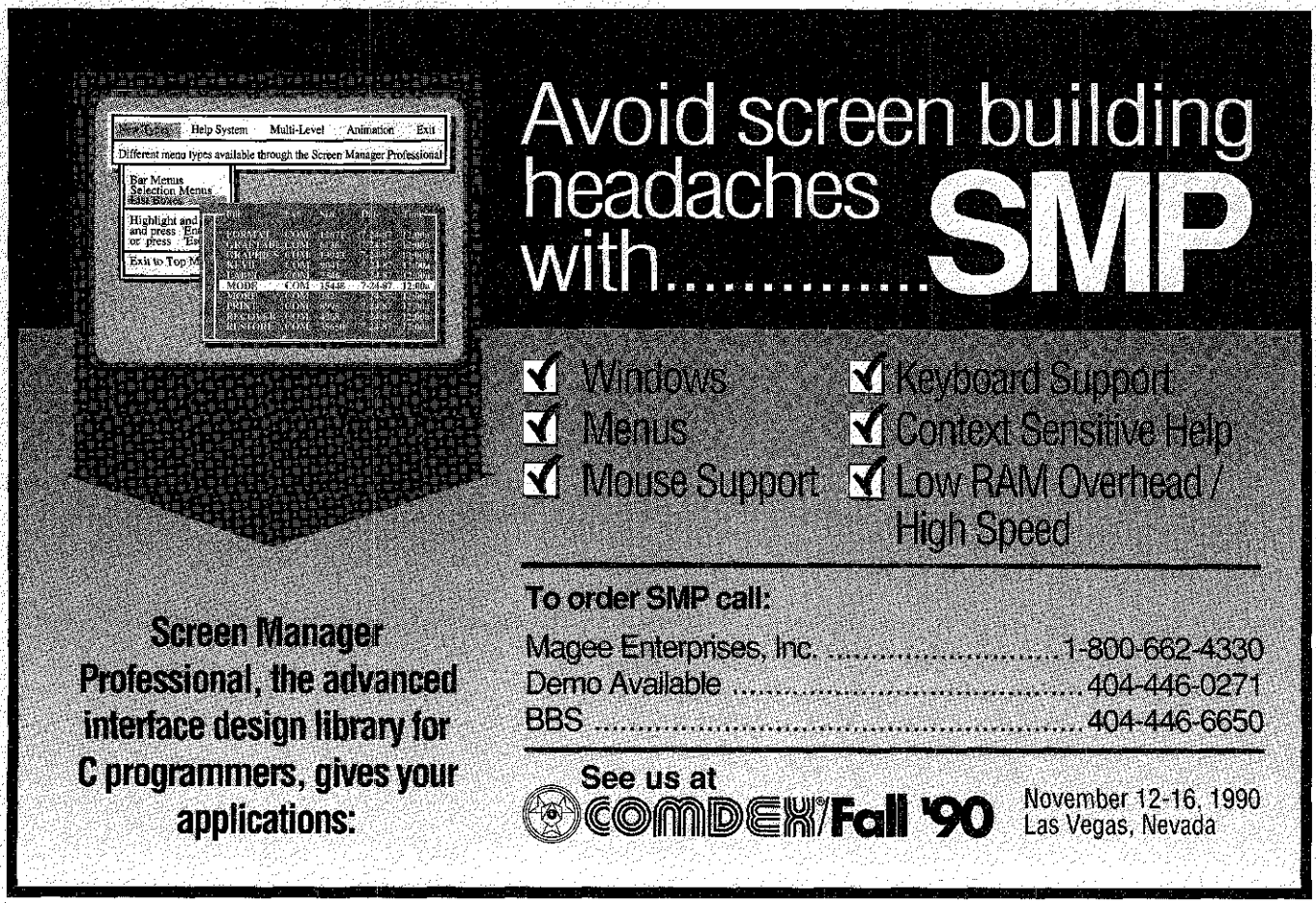
THREE NEW INSTRUCTIONS

The 486 has added three new application instructions: **bswap**, **xadd**, and **cmpxchg**. These instructions are *not* available on the 80386.

The **bswap** instruction reverses the byte order of a 32-bit register, converting a value in little/big endian form to big/little endian form.

```
TEMP          <- r32
r32(7..0)     <- TEMP(31..24)
r32(15..8)    <- TEMP(23..16)
r32(23..16)   <- TEMP(15..8)
r32(31..24)   <- TEMP(7..0)
```

The **xadd** instruction (Exchange and



Circle 246 on Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

ADD) loads DEST into SRC, and then loads the sum of DEST and the original value of SRC into DEST.

```
TEMP    <- DEST
DEST    <- TEMP + SRC
SRC     <- TEMP
```

The **cmpxchg** instruction (Compare and Exchange) compares the accumulator (AL, AX, or EAX register) with DEST. If they are equal, SRC is loaded into DEST. Otherwise, DEST is loaded into the accumulator.

```
IF accumulator = DEST
   ZF <- 1
   DEST <- SRC
ELSE
   ZF <- 0
   accumulator <- DEST
```

Compiler vendors are free to generate these new instructions in situations where it is advantageous. Consider the following C fragment:

```
foo() {
    int temp;
    int x = 3;
    int y = 10;
    ...
    temp = x;
    x = x + y;
    y = temp;
    ...
}
```

Using the new 486 **xadd** instruction, this C program can be translated using only three assembly statements:

```
mov     eax,3
mov     ecx,10
xadd    eax,ecx
```

The disadvantage of using these new instructions is that your application is no longer downwardly compatible with 80x86 processors.

THE CLASSIC QUESTION

The three major 80486 optimizations are the new calling convention, instruction scheduling, and code alignment. Depending on the application, speed improvements on the 486 can range from 5% to 50% when all the optimizations are implemented. Large applications which do very few floating point operations will show very little improvement. For well-defined applications with small procedures that mix integer and floating point operations, large improvements will be noted.

Is buying a compiler tuned for the 486 worth it? That question can only be answered by the reader. I hope the information given here will help you make the best choice for your needs. PJ

People are talking about us.

F77L-EM/32

Port 4GB mainframe programs to 80386s with this 32-bit DOS-extender compiler. The Winner of *PC Magazine's* 1988 Technical Excellence Award just got better. New Version 3.0 and OS include: Editor, Make Utility, Virtual Memory Support, DESQview Support, New Documentation and Free Unlimited Runtime Licenses. F77L-EM/32 $895 OS/386 $395

NEW RELEASE

F77L

The compiler of choice among reviewers and professionals. Includes a Debugger, Editor, Profiler, Linker, Make Utility, Weitek and 386 Real-Mode Support, Graphics. $595

Lahey Personal Fortran 77

New Version 3.0: Full ANSI 77, Debugger, Editor, Linker, Library Manager, Microsoft and Borland C interfaces, 400 page Manual, Unbeatable Price. $99

Contact us to discuss our products and your needs. (800) 548-4778
Lahey Computer Systems, Inc. P.O. Box 6091, Incline Village, NV 89450
Tel: (702) 831-2500 FAX: (702) 831-8123 Tlx: 9102401256
FORTRAN IS OUR FORTE

Circle 44 on Reader Service Card

SUBSCRIBE NOW

FREE ISSUE!

PJ readers are invited to receive a FREE sample issue of Midnight Engineering.

The Journal Of Personal Product Development provides unique mix of practical hardware and software engineering mixed with the business aspects of small scale product development. Your guide to developing and marketing your own products.

Midnight Engineering
Suite 7041
111 E. Drake Rd.
Ft. Collins, CO 80525
303-225-1410 (anytime)

Circle 272 on Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Calendar conversions

Using Julian Day Numbers to keep track of time

By Murray L. Lesser

Perpetual calendar programs can provide the day-of-week for any calendar date, as well as the number of days that have elapsed between any two dates. The basis for these programs is the calculation of a modified Julian Day Number.

The published programs I have seen have all been written in C, and all suffer from two deficiencies. The minor fault is poor design, in that all use an analytical expression requiring floating point arithmetic for computing the day-of-year corresponding to the date in question, even though an iterative procedure requiring only integer arithmetic has been available to C users at least since 1978 [Reference 1].The major fault has to be due to programmers' ignorance, since those programs are based on the erroneous assumption that the entire world switched from the Julian to the Gregorian calendar system on the same day, October 5/15, 1582. This assumption makes the programs useless to genealogists, historians, or anyone else who has to relate near-simultaneous events from separate documents, each dated in its own calendar system.

In this article, I present a module of Julian Day Number calendar functions written for compiled BASIC, using only integer arithmetic. These functions enable you to convert from a calendar date to a Julian Day Number and back again, as well as determine the day of the week for any Julian Day Number.

The functions require the use of long integers, so cannot be used with compiled BASIC prior to QuickBASIC version 4.0. The input (or output) dates can be either any Julian calendar date or a Gregorian calendar date not earlier than October 15, 1582. A demonstration program converts dates between the two calendar systems. I also discuss some basic calendar lore so you will know how to design your own calendar functions in your language of choice.

THE JULIAN DAY NUMBER

The basis of all the calendar conversion routines is the Julian Day Number, an abstraction used to relate astronomical events without getting confused by the local date (or time zone) under any calendar system. Thus it is a useful intermediary reckoning when computing conversions from one calendar system to another. Astronomers count Julian Days in the same way that computer people count memory addresses—starting at zero.

Julian Day 0 started at Greenwich mean noon, January 1, 4713 B.C., Julian proleptic calendar. The count increases by 1 for every 24 hours that has passed since. Decimal fractions are added to indicate portions of the 24-hour period when that level of detail is required.[2]

A major use of the Julian Day concept for non-astronomers is to calculate the time interval in days between two calendar dates. For this purpose, it is convenient to modify the definition somewhat and let each Julian Day start at 12:00 a.m. local civil time instead of at noon GMT, and to use only whole-number values. Using this definition, Tuesday, January 1, 1991 is Julian Day Number 2,448,258—all day, everywhere. A simple algorithm is used to calculate the day of the week from this "civil" Julian Day Number.

The Julian Day Number can be calculated for any calendar system subject to rules establishing the length of each month and year, and having a starting date that can be expressed in terms of the Julian calendar Christian Era. I will consider here only the two major European (and American) systems. The Julian calendar, as we know it today, was defined in 525 A.D. It was gradually superseded by the Gregorian calendar, beginning in 1582. The actual acceptance date of the Gregorian reform varied by country; the Julian calendar was in use in some Eastern European countries well into the first quarter of the twentieth century.

THE JULIAN CALENDAR

The original Julian calendar was started in the Roman Empire by Julius Caesar in 45 B.C. Prior to that time, the Roman calendar was a lunar calendar, with an extra lunar month added by the ruling authority whenever necessary to keep the calendar in step with the solar year. The vernal equinox has religious significance in most societies, so it is desirable to have it fall on a consistent calendar date. Apparently Julius was not overly fussy, and by 46 B.C. he had mismanaged the Roman calendar until it was about two months out of step with the equinox.

Julius Caesar (following the advice of Alexandrian astronomer Sosigenes)

Murray L. Lesser is a retired IBM engineer. Murray has written three books dealing with enhancing various Microsoft BASIC compilers with assembled modules. The most recent of these, Advanced QuickBASIC 4.0, *was published by Bantam.*

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

threw enough extra lunar months into the year 46 B.C. to increase its length to 445 days, and inaugurated **his** algorithmic calendar with the year 45 B.C. In the "Julian" calendar, the mean length of the year is 365.25 days. Most years have 365 days, but an intercalary day is inserted every fourth year. Julius Caesar didn't know he was living in B.C. In his time, the Roman calendar started at the legendary founding of the city—corresponding to the Julian proleptic calendar year 753 B.C.

The next major change was the designation of the Christian Era. In 525 A.D., the Roman abbot Dionysius Exiguus was extending the table that was used to calculate when Easter would fall. While he was at it, he changed the year number of the (then current) Diocletian era to agree with what is now known as the Christian Era, beginning in the year 1 A.D.

Calendar eras do not have a year 0. Hence, the year prior to 1 A.D. is 1 B.C., which is a leap year. For astronomical and calendar calculation purposes, 1 B.C. is considered to be the year 0, and all remaining B.C. years are negative, with the absolute value of a negative astronomical year being one less than the B.C. year (e.g., 2 B.C. is astronomical year -1). Any astronomical year number exactly divisible by 4 is a leap year in the Julian calendar.

Since the first year of the Christian Era is 1 A.D., the first century of this era ended on the last day of the year 100. Thus, the second century started with the year 101. In spite of what you may have read in the popular press, the twenty-first century will not start until January 1, 2001.

While the abbot's system rapidly spread throughout Christendom, there was no general agreement as to when each new year started. As each locality adopted the Christian Era calendar, it seemed to have picked its own New Year's Day. This is unfortunate, because there is no way to precisely "date" many events recorded in the Julian calendar without knowing the local New Year's day at the time.

When England adopted the Christian Era, its official new year began on what we would now consider to be the previous December 25. England set March 25 as the beginning of the calendar year in the 14th century, and used that date until the Gregorian calendar was adopted. By 1700, it had become the custom both in England and in its American colonies to use double-year dating for all documented events from January 1 through March 24.[3] Therefore, when George Washington was born, his birth would have been noted in the family Bible as February 11, 1731/2. When England adopted the Gregorian calendar, George's birth date was converted to the equivalent February 22, 1732.

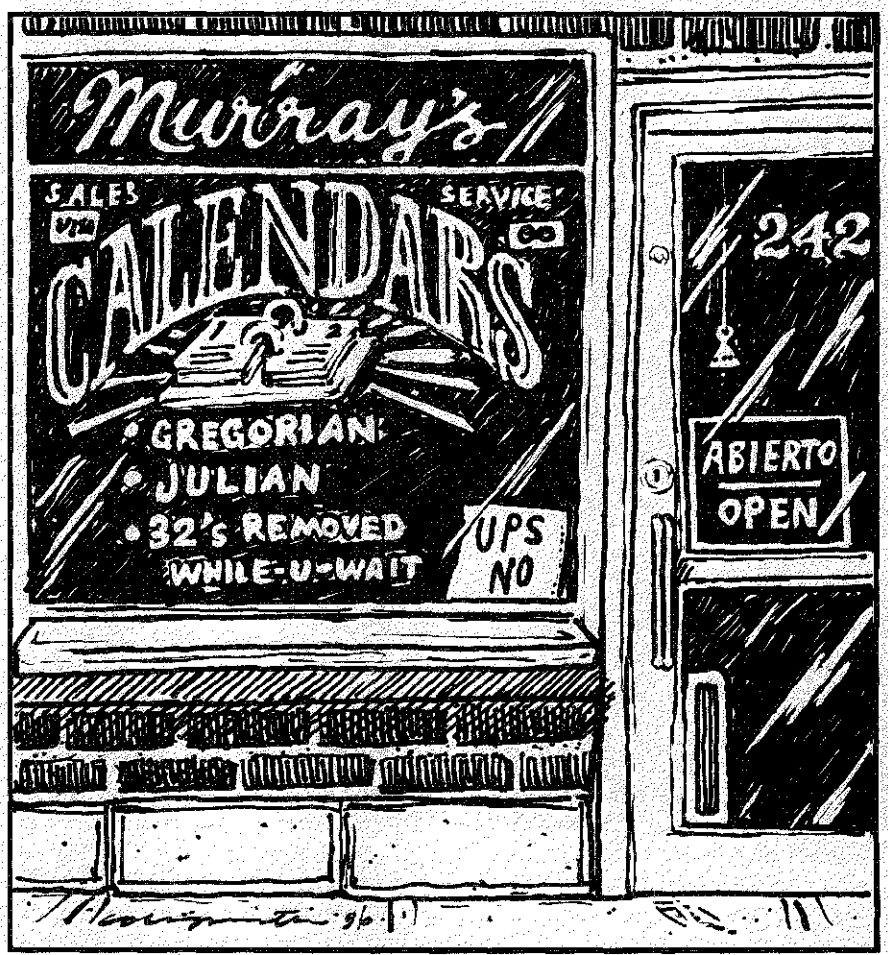
THE GREGORIAN REFORM

The Julian calendar has 36,525 days per 100 years. However, 100 solar years have only about 36,524.22 days. I use "about" because the actual number of days between vernal equinoxes changes slightly with time. The year now is running between one and two seconds long.

As you can see from these numbers, the Julian calendar gains a little over three days on the sun every 400 years. By the 16th century, the actual vernal equinox was falling on March 11, rather than on the March 21 used in calculating the date of Easter, and Easter was moving toward the beginning of summer. Consequently, Pope Gregory XIII promulgated the Gregorian calendar reform in 1582.

The reform consisted of deleting the unwanted 10 days; the day after Thursday, October 4, 1582 was designated to be Friday, October 15. The extra three days per 400 years were eliminated by designating years evenly divisible by 100 as leap years only if they were also divisible by 400. This way, 100 Gregorian years have only 36,524.25 days, which is very close to the solar calendar. It will take a little over 3300 years before the Gregorian calendar gains a day on the sun.

The Gregorian calendar was adopted fairly rapidly in countries in which Roman Catholicism was the state religion. It came into use much more slowly in Protestant countries, and not until the twentieth century in Greek Orthodox



PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

FIGURE 1

```
C>CONVERT 1/1/3268

The Julian calendar date,
     January 1, 3268, is a Monday
The Julian Day Number is 2,914,695
The equivalent Gregorian calendar
     date is January 23, 3268
```

countries. Great Britain and its Dominions adopted the Gregorian calendar by act of Parliament, effective September 3/14, 1752. The beginning of the official year was changed to January 1 at the same time.

THE JULIAN PERIOD

The chronological measure of time known as the Julian Period was introduced in the 16th century by the Italian scholar Josephus Justus Scaliger (1540-1609). Although the period is based on the Julian year, it is not named for the original Julius Caesar, but rather for Scaliger's father, Julius Caesar Scaliger (1484-1558).

The definition of the Julian Day Number is most certainly not attributable to Scaliger. For one thing, the Greenwich Royal Observatory was not founded until 1675. For another, scholars in Scaliger's time didn't start their counts at 0, they started at 1.

The Julian Period is 7980 Julian calendar years, which is the least common multiple of the 19-year lunar cycle, the 28-year solar cycle, and the 15-year non-astronomical cycle of the indiction. The epoch of the Julian Period is 4713 B.C., Julian proleptic calendar. Obviously, Scaliger chose 4713 B.C. as Julian Period year 1 because that was the most recent year in which all three cycles were also in year 1. Astronomers use 4713 B.C. today because it predates all historical events.

The lunar cycle is the 19-year cycle designating fixed tables of the "ecclesiastical" dates of the new moon for the Julian calendar, used primarily in establishing the date of Easter in that system. The solar cycle represents the 28-year interval after which the days of the week recur on the same days of the Julian year. While the origins of the 15-year cycle of indiction are unclear, the cycle was well-established by the fourth century A.D. and appears to be related to tax accounts.

THE CALENDAR ROUTINES

The BASIC functions that do all the calendar-conversion work are in the module JULIAN.BAS (Listing 1). The Julian Day Number is calculated from the input date in the **julian** function. For the Julian calendar, the number of whole years that have elapsed since the start of the Julian Period is counted, effectively multiplied by 365.25, and then the number of elapsed days in the "current" year is added to the product.

It is a little more difficult to calculate the Julian Day Number for dates in the Gregorian calendar (indicated by a "G" input *date.code* byte). I have calculated the number of elapsed Gregorian calendar days from a "fictitious" Gregorian origin of January 1, 0, which would have been a leap year in the Gregorian proleptic calendar. After calculating the number of days that have elapsed since this origin, based on Julian-calendar years, a correction is made for extra days that would have been leap days in the Julian calendar but were not in the Gregorian. The Julian Day Number of the origin is added to this result.

My fictitious Gregorian calendar starting date is actually January 3, 1 B.C. (Julian), which is Julian Day Number 1,721,060. The program returns an error signal instead of a Julian Day Number for any Gregorian date prior to October 15, 1582. I used the input convention that Julian calendar B.C. dates are indicated as negative, with any dates in year "0" producing an error return. This is not the best possible arrangement, since such dates might be confused with astronomical years.

The function **calendar** reverses the process, calculating the calendar date corresponding to a given Julian Day Number. The function **weekday** returns a coded day-of-week (0 for Sunday, 1 for Monday, etc.) corresponding to a given Julian Day Number. Both of these functions return error indications if they are given an invalid input.

CONVERT.BAS (Listing 2) is a small demonstration program. In its default mode, it converts a Julian calendar date to Gregorian, giving the day-of-week and the Julian Day Number in passing. Gregorian dates prior to October 15, 1582 are declared invalid. An invalid input or output date will initiate a jump to the error exit at the **oops** label, causing an exit with an *Errorlevel* signal.

JULIAN.BAS can be compiled with any Microsoft BASIC compiler starting with QuickBASIC's BC 4.0. As shown, CONVERT.BAS must be compiled with BC 7.0 (or later), since the earlier versions did not include the ability to emit an exit code on **end**. To compile CONVERT with BC versions 4.x or 6.0, either move the **oops** label to before the first **end** statement and delete the second one, or else substitute call quit (the assembled subroutine shown in Reference 4) for the second **end**. Compiling with BC 7.0 and linking to the stub file NOFLTIN.OBJ will produce a shorter executable file (and require less code space) than will compiling with earlier versions of BC. This stub file prevents unnecessary floating point routines from being linked in automatically to support the **val** statement.

One warning: although the Julian Day Number for Julian date January 1, 3268 probably should start over with 0, since it begins a new period, I have assumed that the astronomers of the time will find it convenient not to restart the count. If I turn out to be wrong, you can always correct the program. PJ

REFERENCES

1. Brian W. Kernighan & Dennis M. Ritchie, *The C Programming Language*, Prentice Hall, Inc. Englewood Cliffs, 1978, p. 104.

2. *Explanatory Supplement to the Astronomical Ephemeris and the American Ephemeris and Nautical Almanac*, Her Majesty's Stationery Office, London, 1974 [revised] reprint, p. 71. Chapter 14 of this reference is the source of the calendar lore in this article unless otherwise attributed. The book is out of print.

3. Val D. Greenwood, *The Researcher's Guide to American Genealogy*, The Genealogical Publishing Co., Baltimore, 1973, p. 37.

4. Murray L. Lesser, "Exit Codes for QuickBASIC," *Programmer's Journal* 7.4, pp. 86-87.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

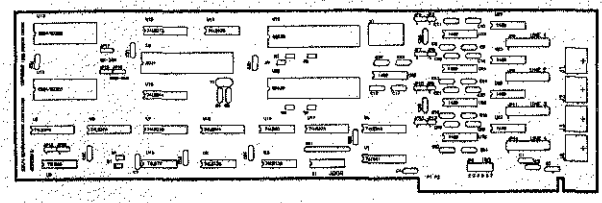
LISTING 1

```
'   JULIAN.BAS - A module of calendar functions.
'    JULIAN() converts calendar dates to the Julian Day Number.
'   CALENDAR() converts a Julian Day Number to a calendar date.  Both
'   functions deal with both Julian and Gregorian calendar dates.
'    WEEKDAY() returns a day-of-week code for any Julian Day Number.
'    Julian proleptic dates earlier than January 1, 4713 B.C. are
'   invalid, as are Julian dates in the year 0.  Gregorian dates prior
'   to October 15, 1582 are also invalid, as are any dates with month
'   numbers not in the range 1 - 12, or with day numbers less than zero
'   or greater than the number of days in that month.  Invalid dates
'   will cause an error return.
'    B.C. Julian dates are denoted by negative year numbers.
'            BASIC version written by M. L. Lesser, 6/8/90
'                Compiled with Microsoft BC version 7.0
'*************************************************************************
    DEFINT A-Z
    DEFLNG J, O
    TYPE DATES
       MONTH AS INTEGER
       DAY   AS INTEGER
       YEAR  AS INTEGER
       CODE  AS STRING * 1
    END TYPE
    OPTION BASE 1                       'Arrays start at 1
    DIM SHARED M.DAYS(12)               'Array of days in each month

'*************************************************************************
'   INIT [DECLARE SUB INIT()] is a subroutine to initialize the M.DAYS
' array. A program using the functions here must call INIT first.
'*************************************************************************
SUB INIT STATIC
    LET M.DAYS(1)  = 31           'February initialized later
    LET M.DAYS(3)  = 31  :  LET M.DAYS(4)  = 30
    LET M.DAYS(5)  = 31  :  LET M.DAYS(6)  = 30
    LET M.DAYS(7)  = 31  :  LET M.DAYS(8)  = 31
    LET M.DAYS(9)  = 30  :  LET M.DAYS(10) = 31
    LET M.DAYS(11) = 30  :  LET M.DAYS(12) = 31
END SUB

'*************************************************************************
'   JULIAN(<date>) [DECLARE FUNCTION JULIAN& (D AS DATES)] returns the
' Julian Day Number corresponding to the input calendar date, or -1&
' if given an invalid input date.  A code flag of "G" denotes input as
' a Gregorian calendar date.  All other code flags denote Julian date.
'*************************************************************************
FUNCTION JULIAN&(DATE AS DATES) STATIC
    IF DATE.CODE = "G" THEN             'Gregorian calculations based on
       LET YEAR = DATE.YEAR             ' a proleptic starting year 0
    ELSE                                'Julian Dates may have B.C. (-)
       IF DATE.YEAR = 0 THEN GOTO ERR.EXIT  ' years, but no year 0.
       IF DATE.YEAR < 0 THEN                'Convert B.C. years to
          YEAR = DATE.YEAR + 1              ' astronomical date
       ELSE
          YEAR = DATE.YEAR
       END IF
       YEAR = YEAR + 4712               'Absolute years in Julian Period
    END IF
' Check for invalid input dates:
    IF (YEAR < 0) OR (DATE.MONTH < 1) OR (DATE.MONTH > 12) OR _
      (DATE.DAY < 1) THEN GOTO ERR.EXIT
' Fill in February array value after testing for leap YEAR:
' (LEAP flag set to -1 if true, else to 0)
    LET LEAP = (YEAR MOD 4 = 0)
    IF (DATE.CODE = "G") THEN LET LEAP = LEAP AND ((YEAR MOD 100 <> 0) _
      OR (YEAR MOD 400 = 0))            'Flag reset for Gregorian
    IF LEAP THEN LET M.DAYS(2) = 29 ELSE M.DAYS(2) = 28
' Check for valid day of month:
    IF DATE.DAY > M.DAYS(DATE.MONTH) THEN GOTO ERR.EXIT
' Calculate day of year:
    LET DAY.YEAR = DATE.DAY
    FOR I = 1 TO DATE.MONTH - 1
       LET DAY.YEAR = DAY.YEAR + M.DAYS(I)
    NEXT I
' Calculate Julian Day Number for Julian calendar:
    JUL.DAY = (365& * YEAR) + DAY.YEAR + (YEAR-1)\4
    IF YEAR = 0 THEN LET JUL.DAY = JUL.DAY - 1  'First JUL.DAY is zero
' If for Gregorian calendar, correct for fewer leap days:
    IF DATE.CODE = "G" THEN
       JUL.DAY = JUL.DAY + (YEAR-1)\400 - (YEAR-1)\100 + 1721060&
       IF JUL.DAY < 2299161& THEN GOTO ERR.EXIT         '10/15/1582
    END IF
    JULIAN = JUL.DAY
EXIT FUNCTION           'Normal function exit
```

continued

Intelligent Four Channel

Serial Communications Controller



- 8031 microcontroller
- 82530 SCCs
- 32K memory
- RS-232
- RJ-45 connectors
- DOS device driver
- FCC Class A approved
- Excellent documentation
- BASIC, C and Pascal software source examples
- **$495 complete** with all cables, software, & extensive documentation

Designed, Manufactured, Sold and Serviced by:

Rtech Inc.

Voice . . . (612) 345-4555
FAX . . . . (612) 345-5514
Modem . . . (612) 345-4656

907 N. 6th St., Lake City, MN 55041

Circle **36** on Reader Service Card

Real-Time Multitasking

Do the multitasking systems you are considering using seem a bit like overkill or need an awful lot of memory? Maybe all you really need is a system which can run more than one DOS program concurrently. If this sounds like your situation, consider:

MultiDos Plus™

Flexible preemptive scheduling of multiple tasks with full access to MS-DOS or PC-DOS. Supports the loading and execution of tasks in LIM 4.0 expanded memory. For more information please call for a free brochure.

Call our BBS for free demo software.

The complete software package is only $99 + S/H. Visa/MasterCard accepted.

30-day money back guarantee.
(subject to a $15 restocking fee)

Nanosoft Inc.
13 Westfield Road
Natick, MA 01760
800-678-2141
508-651-0091
FAX 508-655-8860
BBS 508-650-9552

Circle **109** on Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

```
ERR.EXIT:
    LET JULIAN = -1&
END FUNCTION            'Function exit for invalid input date

'*********************************************************************
'   CALENDAR(<jul.day>,<date>) [DECLARE CALENDAR% (J&, D AS DATES)]
' returns a -1 (true) if it can calculate a valid output date from the
' Julian Day Number argument, or a 0 (false) if it can not.  If valid,
' the calculated date is stored in the DATES record named in the
' argument list.  On input, only the CODE byte of the DATES record is
' used to specify which calendar will be used for output.
'*********************************************************************
FUNCTION CALENDAR% (JUL.DAY,DATE AS DATES) STATIC
    IF DATE.CODE = "G" THEN
        IF JUL.DAY < 2299161& THEN GOTO OOPS    'Invalid Gregorian date
        LET OFFSET = JUL.DAY - 1721060& 'Gregorian proleptic start date
    ELSE                                'Output Julian date
        LET OFFSET = JUL.DAY
    END IF
    IF OFFSET < 0 THEN GOTO OOPS        'Invalid Julian date
' Calculate (absolute) year and remaining days for Julian calendar:
    YEAR = (OFFSET * 4)\1461            '1461/4 = 365.25
    DAY.YEAR = OFFSET - (365 * YEAR) - (YEAR-1)\4
    IF OFFSET <= 365& THEN DAY.YEAR = DAY.YEAR + 1  'Correct for year 0
    LEAP = (YEAR MOD 4 = 0)
' If Gregorian, correct current date for extra Julian leap years:
    IF DATE.CODE = "G" THEN
        DAY.YEAR = DAY.YEAR + (YEAR-1)\100 - (YEAR-1)\400
        LEAP = LEAP AND ((YEAR MOD 100 <> 0) OR (YEAR MOD 400 = 0))
        IF DAY.YEAR > 365 - LEAP THEN
            LET YEAR = YEAR + 1
            LET DAY.YEAR = DAY.YEAR - 365 - LEAP
            LET LEAP = (YEAR MOD 4 = 0) AND ((YEAR MOD 100 <> 0) OR _
                (YEAR MOD 400 = 0))         'Recalculate
            IF LEAP THEN LET M.DAYS(2) = 29 ELSE M.DAYS(2) = 28
        END IF
    END IF
' Calculate calendar date:
    IF LEAP THEN LET M.DAYS(2) = 29 ELSE M.DAYS(2) = 28
    LET I = 0
    WHILE DAY.YEAR > 0
        LET I = I + 1
        LET DAY.YEAR = DAY.YEAR - M.DAYS(I)
    WEND
    LET DATE.MONTH = I
    LET DATE.DAY = DAY.YEAR + M.DAYS(I)
    IF DATE.CODE = "G" THEN
        LET DATE.YEAR = YEAR
    ELSE
        LET DATE.YEAR = YEAR - 4712
    END IF
' If B.C., correct for astronomical year:
    IF DATE.YEAR < 1 THEN LET DATE.YEAR = DATE.YEAR - 1
    LET CALENDAR = -1
EXIT FUNCTION
OOPS:
    LET CALENDAR = 0
END FUNCTION

'*********************************************************************
'   WEEKDAY(<jul.day>) [DECLARE FUNCTION WEEKDAY% (J&)] returns a day-
' of-week code (0 = Sunday, 1 = Monday, etc.) if given a valid Julian
' Day Number, or -1 if the Julian Day Number is invalid.
'*********************************************************************
FUNCTION WEEKDAY%(JUL.DAY) STATIC
    IF JUL.DAY >= 0 THEN
        WEEKDAY = (JUL.DAY + 1) MOD 7
    ELSE
        WEEKDAY = -1
    END IF
END FUNCTION
```

LISTING 2

```
' CONVERT.BAS - A test driver for the calendar functions in JULIAN.BAS *
'                                                                      *
'    CONVERT accepts an input date in the form mm/dd/yyyy[/G], where
' the optional [/G] denotes that the input date is for the Gregorian
' calendar.  If the input date is Julian (default), the day of the
' week and the Julian Day Number are displayed; if the date is later
' than 10/5/1582, the corresponding Gregorian date will also be
' displayed.  Negative values of yyyy denote B.C. Julian dates.
'    An input Gregorian date will return the day of week, the Julian
' Day, and the corresponding Julian date.  Gregorian dates prior to
' 10/15/1582 are not accepted.
'           BASIC version written by M. L. Lesser, 6/9/90
'            Compiled with Microsoft BC 7.0, switch "/O"
'          Linked to JULIAN and NOFLTIN, switches "/NOE/E"
'**********************************************************************
    DEFINT A-Z
    DEFSTR C, N, T
    DEFLNG J
  TYPE DATES
      MONTH AS INTEGER
      DAY   AS INTEGER
      YEAR  AS INTEGER
      CODE  AS STRING * 1
  END TYPE
    DECLARE SUB INIT ()
    DECLARE FUNCTION JULIAN& (D AS DATES)
    DECLARE FUNCTION CALENDAR% (J&, D AS DATES)
    DECLARE FUNCTION WEEKDAY% (J&)
    DIM INDATE AS DATES, OUTDATE AS DATES
    DIM SHARED NAME.M(12), NAME.D(6)

' Local subroutine to display dates as strings:
SUB SHOWDATE (D AS DATES) STATIC
    PRINT NAME.M(D.MONTH) STR$(D.DAY) "," STR$(ABS(D.YEAR));
    IF D.YEAR < 0 THEN PRINT " B.C.";
END SUB
' Initialize arrays:
    LET NAME.M(1)  = "January" :   LET NAME.M(2)  = "February"
    LET NAME.M(3)  = "March"   :   LET NAME.M(4)  = "April"
    LET NAME.M(5)  = "May"     :   LET NAME.M(6)  = "June"
    LET NAME.M(7)  = "July"    :   LET NAME.M(8)  = "August"
    LET NAME.M(9)  = "September":  LET NAME.M(10) = "October"
    LET NAME.M(11) = "November":   LET NAME.M(12) = "December"
    LET NAME.D(0) = "Sunday"   :   LET NAME.D(1) = "Monday"
    LET NAME.D(2) = "Tuesday"  :   LET NAME.D(3) = "Wednesday"
    LET NAME.D(4) = "Thursday" :   LET NAME.D(5) = "Friday"
    LET NAME.D(6) = "Saturday"
    CALL INIT                   'Initialize M.DAYS array in JULIAN
' Get input date from the command line:
    LET TEXT = COMMAND$
    IF LEN(TEXT) = 0 THEN
        PRINT: PRINT "Syntax:  CONVERT mm/dd/yyyy[/G]"
        PRINT "          (B.C. years are negative - Use ";
        PRINT CHR$(34) "/G" CHR$(34) " switch for Gregorian calendar)"
        GOTO OOPS               'Error exit
    END IF
    LET INDATE.MONTH = VAL(TEXT)
    LET TEXT = MID$(TEXT,INSTR(TEXT,"/")+1)
    LET INDATE.DAY = VAL(TEXT)
    LET TEXT = MID$(TEXT,INSTR(TEXT,"/")+1)
    LET INDATE.YEAR = VAL(TEXT)
    LET I = INSTR(TEXT,"/")
    IF I <> 0 AND LEN(TEXT) > I THEN
        LET INDATE.CODE = UCASE$(MID$(TEXT,I+1,1))
    ELSE
        LET INDATE.CODE = "J"
    END IF
    LET CALEND = "Julian"
    IF INDATE.CODE = "G" THEN LET CALEND = "Gregorian"
    LET JUL.DAY = JULIAN(INDATE)
    PRINT: PRINT "The " CALEND " calendar date, ";
    CALL SHOWDATE(INDATE)
    IF JUL.DAY < 0 THEN
        PRINT ", is an invalid input date."
        GOTO OOPS
    ELSE
        PRINT ", is a " NAME.D(WEEKDAY(JUL.DAY))
    END IF
    PRINT "  The Julian Day Number is" USING " #####,###"; JUL.DAY
    IF LEFT$(CALEND,1) = "J" THEN
        LET CALEND = "Gregorian"
    ELSE
        LET CALEND = "Julian"
    END IF
    LET OUTDATE.CODE = LEFT$(CALEND,1)
    PRINT "  The equivalent " CALEND " calendar date is ";
    IF CALENDAR(JUL.DAY,OUTDATE) THEN
        CALL SHOWDATE(OUTDATE)
    ELSE
        PRINT "an invalid date"
    END IF
    PRINT
END                     'Normal exit
OOPS:
END 1                   'Error exit (returns Errorlevel 1)
```


PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Software components

Designing generalized components by creating abstractions

By Richard Riehle

Software reuse has the same advantage as theft over honest toil.
—Thomas Standish, 1983

At the 1969 NATO conference on Software Engineering, when Doug McIlroy presented his "Mass Produced Software Components" paper, the concept was lauded as one of the great new ideas for managing the software development process. Now, over twenty years later, software gurus are still in agreement about the concept; however, the software industry has barely started implementing applications with predefined components.

Some big gains in the direction of uniform software have been made in spite of the industry's delayed response. Development tools such as dBASE and Paradox could be regarded as high-level instances of generalized software components. Libraries of mathematics routines and generic data structures are available in the marketplace. The standard statistical routines in the Biomedical (BMSS) package are used by every "cookbook" statistician working on an advanced degree in the social sciences. C and Pascal compilers have **include** files for a wide variety of services and device drivers. Dynamic calls to external subroutines are common for many language environments. And it is rare for any applications programmer to write the detail code for sorting a file.

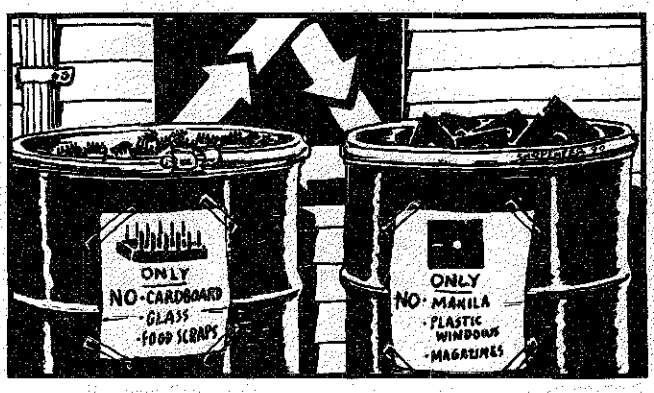
In spite of these gains, there is still a long way to go. In the past ten years we have heard more and more about the concepts of "software IC's," loose coupling, modularity, and designing through raising the level of abstraction. We have been intellectually pummelled by advocates of the expression *object-oriented*. We have object-oriented design, object-oriented programming, object-oriented requirements analysis, and object-oriented databases. Soon I expect to see a reference to object-oriented functionally decomposed entity-relationship hierarchical state-transition diagram development methods.

The term object-oriented seems to have been co-opted by the inheritance people. It is not clear that inheritance is of universal benefit to component-oriented programming, especially if one insists on coupling inheritance with dynamic binding. I have decided to suppress my devotion to the term *object-oriented* when discussing component-oriented software engineering. In my mind, object-oriented/inheritance software development is just another subset of component-oriented software engineering.

The other term that gets a lot of attention is *reuse*. Unfortunately, the careful placement of an "f" in *reuse* will aptly describe the quality of much of the software that claims to be reusable. My preferred term for such software would be *generic*, but that is an Ada-reserved word, and will be a tough sell in Smalltalkland. Instead, I will use the term *generalized*.

COMPONENT DESIGN

For this article, a component is a pre-designed unit of software with predictable behavior that can be incorporated into a larger software design without modification. Such a component



Richard Riehle is a software engineer with AdaWorks, 1090-105 Tanland Drive, Palo Alto, CA 94303, (415) 858-1551.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

should be designed independently of a hardware or operating system platform. It may be a generalized template that is independent of specific data types or an abstract data type. A component differs from code leveraging in one important respect. The generalized nature of the component is deliberate. While code leveraging is based on the discovery of currently existing reusable code, components are based on planning for on-going systems engineering. Both may have salubrious results in a final system implementation, but one is based on planning ahead and the other on serendipity.

Components can be designed for the dynamic binding environments typified by object-oriented programming languages (Smalltalk, C++, Objective C) or for static binding languages such as Ada. Both environments are suitable. The language decision, therefore, will vary according to what is most appropriate for developing the required application.

What part of the component is generalized? Programming is one of the last careers in which someone can be paid for reinventing the wheel. In his book on algorithms (*The Art of Computer Programming*, Vol. 3), Donald Knuth says that nearly all programming tasks can be classed as some kind of sorting or searching activity. In *Algorithms + Data Structures = Programs*, Niklaus Wirth describes programs as "concrete formulations of abstract algorithms based on particular representations and structures of data." We are able to generalize components by creating abstractions for algorithms and data structures. For example, if an algorithm can be generalized to the point where it can be used with any type of data, it can then be coded in a high-level language and used over and over again in engineered software.

The objective is to raise the level of abstraction of a software component by encapsulating the algorithm(s) within a cohesive programming unit such as a module (Modula 2), package (Ada), or class (C++). Whenever possible, the component algorithm will be coded so the data type will be resolved at a later stage of binding—during compilation within the client program, during linkage, or dynamically. C++, Smalltalk, and Ada provide this capability in varying degrees. In some cases the algorithm will implement an instance of an abstract data type (ADT). An ADT may be unique to the algorithm, while the component may still be generalized.

COMPONENT EXAMPLES

A component that is not type-specific can be incorporated into a wide range of software solutions. An Ada example is:

```
generic
      type Element is private;
procedure SWAP (LEFT, RIGHT : in out Element) is
      WORK_AREA : Element;
begin
      WORK_AREA := LEFT;
      LEFT      := RIGHT;
      RIGHT     := WORK_AREA;
end SWAP;
```

This trivial example, procedure **swap**, could now be used for any data type for which assignment (:=) is pre-defined, including integers, floating point, fixed point, and records. This procedure might be used in the implementation of a bubble sort. Similar capabilities are available for C++ and other object-oriented programming languages.

A software unit such as the **swap** procedure represents only one aspect of component design. The "abstract data type" mentioned above is represented as a type whose associated operations are encapsulated into a single component. Ideally, all the legal operations on the ADT are explicitly revealed in the visible part of the component. This would be the definition (Modula-2), specification (Ada), or public (C++) part of that unit. The visible part is the "interface" to the facilities of the component. It contains everything a programmer needs to use the component. Remember, a component must be usable without modification. Therefore, the programmer is not required to know the underlying algorithm that supports the interface specification. This can be described as *information hiding*, a term used originally by David L. Parnas.

Using Ada again:

```
with NON_ZERO;
package RATIONAL_NUMBERS is
    type FRACTION is private;

    function ">" (LEFT,RIGHT : FRACTION) return BOOLEAN;
    function "+" (LEFT,RIGHT : FRACTION) return FRACTION;
    function "-" (LEFT,RIGHT : FRACTION) return FRACTION;
    function "*" (LEFT,RIGHT : FRACTION) return FRACTION;
    function "/" (LEFT,RIGHT : FRACTION) return FRACTION;
    ...[More operations would usually be defined]...
  private
     type FRACTION is record
          NUMERATOR   : NON_ZERO.NON_ZERO_INTEGER;
          DENOMINATOR : NON_ZERO.NON_ZERO_INTEGER;
  end RATIONAL_NUMBERS;
```

Here, the available operations are defined in the visible part of the package and the full definition of the type in the private part of the specification. Any program using **RATIONAL_NUMBERS** could define a type of fraction. For example:

```
F1, F2, F3   : RATIONAL_NUMBERS.FRACTION;
NON_FRACTION : POSITIVE;
```

That program could then include statements such as:

```
if F1 = F2 then ... end if;
if F2 > F3 then ... end if;

F3 := F1 + F2;
F3 := F1 * F2;
```

but would not be able to perform:

```
if  F1 < F3 then ... end if;      -- illegal
if  F1 >= F3 then ... end if;     -- illegal

F1 := F3**F2;                     -- illegal
F1 := F3 + NON_FRACTION;          -- illegal
```

The package imports another abstract data type called **NON_ZERO_INTEGER** from package **NON_ZERO**. The only operations permitted on objects of type **FRACTION** are those shown in the package specification. A client package/subprogram of **RATIONAL_NUMBERS** is denied access to the **NUMERATOR** or **DENOMINATOR** shown in the private part of the package.

The algorithmic part of the package is coded within the non-visible part, the package body. In Ada and Modula-2 (implementation module), this is typically a separate compilation unit.

Although **RATIONAL_NUMBERS** provides a well-defined component for the abstract data type **FRACTION**, it is still not as generalized as we might wish. It is often appropriate for an engineered component to represent a specific type.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

However, we want a more generalized abstract data type that will implement a set of operations in the form of a template. Data structures are well known, and there is wide acceptance for many of their corresponding algorithms. Common data structures—e.g., lists, rings, trees, maps, and matrices—fit nicely into the component domain for generalized templates.

An entire industry has developed around this concept in the Ada world. One company, EVB Software Engineering in Frederick, Maryland, sells a library of almost 300 engineered components. That library represents nearly every data structure any reasonable person could want. Other companies are emerging with similar libraries for C++. The EVB components have the benefit of a standardized language environment (Ada) so they are not plagued by the poison in the well resulting from varying implementations of their language environment.

An example of such a component is the common *first in, first out* (FIFO) queue. As an Ada generic component, a partial implementation of the specification might look like this:

```
generic
     type ITEM is private;
package FIFO_QUEUE is
     type LIST is limited private;

     procedure PUT  (THE_ITEM : in ITEM;
                     THE_LIST : in out LIST);

     procedure GET  (THE_ITEM : out ITEM;
                     THE_LIST : in out LIST);

     procedure COPY (SOURCE_LIST : in LIST;
                     TARGET_LIST : in out LIST);

     function IS_EMPTY (THE_LIST : LIST ) return BOOLEAN;

     ...[Many more operations would be required]...

private
        -- full definition of the type LIST
end FIFO_QUEUE;
```

This abstract data type, **LIST**, may be implemented as a bounded list (usually an array), an unbounded list (using pointers), and so on. The client of package **FIFO_LIST** only knows the operations presented. There is no need for external access to the representation of objects of type **LIST**.

The generic formal parameter, **ITEM**, can be thought of as the template element. It can be filled in with any type that permits assignment (:=) or tests for equality (=, or /=).

RATED COMPONENTS

When components are engineered as software templates for general use, they begin to enter the realm of what I like to call *rated* components. In the hardware engineering world, an electrical engineer can consult a catalog to determine the rated behavior of a particular capacitor, resistor, or other component. That engineer does not want any surprises. If the behavior of a hardware component is correctly described, anyone incorporating that component into a larger design will find that it operates as rated.

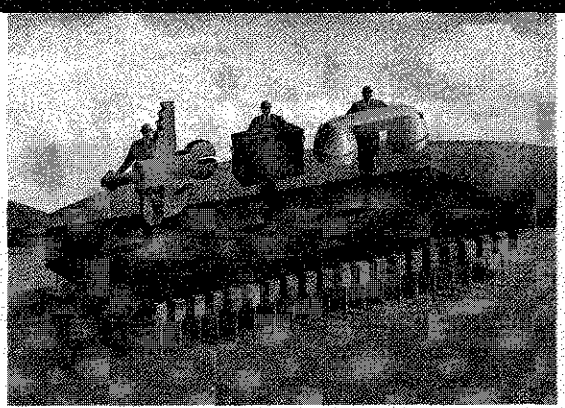
Software designers may eventually require rated components. For component-oriented software engineering to become a reality, such rating should, at minimum, establish that a component will consistently behave as advertised. There may even be an opportunity here for someone to start a new software rating service similar to the famous Underwriters Laboratories. Rated components would be written in high-level languages and be portable, reliable, and contribute to the productivity of other software engineering organizations.

COMPONENT MANAGEMENT

One of the significant difficulties with component-oriented software engineering is the age-old problem of library management. If we cannot find a component, we cannot use it. In fact, the software industry's failure to establish a standard for component designation is a major reason why there is so little use of standard components in most organizations. It may be that there is a need for some organization such as the Association for Computing Machinery to establish a special interest group (SIG) on software nomenclature. Such a nomenclature would consist of primitives that could be used consistently to catalog software components independently of any language implementation.

Of equal importance, programmers and software engineers must be educated to understand how to use such standard components. For example, even if there were a standard nomenclature for data structures, very few programmers are educated to understand the fundamentals of such structures. Rare is the programmer who knows when to use a bounded list versus an unbounded list, or when to use a tree structure versus a doubly-linked ring. As long as programmers are ignorant of fundamental algorithms, data structures, and how to

INTEL DEVELOPMENT TOOLS



For the ultimate success with your microcomputer development project Intel offers you the total solution. From emulators to compilers to performance analysis and simulation, Intel can supply you with the most up-to-date and powerful development tools available today.

We now offer you an easy method of ordering. Simply call our toll-free number at 1-800-874-6835 to request a complimentary copy of our Intel Development Tools Catalog listing all our offerings. Or drop us a line at: Intel Corporation, Catalog Sales, 5200 NE Elam Young Pkwy, JF1-15, Hillsboro, OR 97124. After you have had a chance to look over our products, give us a call. We can answer any questions you may have about Intel Development Tools and we'll be happy to take your order over the phone. No waiting. It's as simple as that. So call us today. We'll be waiting to hear from you.

CALL US TODAY 1-800-874-6835

Circle 50 on Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

design with standard components, then the ideal of component-oriented software engineering will be no more than that: an ideal.

PAST, PRESENT, FUTURE

In early incarnations of COBOL and FORTRAN, the concept of component-oriented software engineering was at best difficult. The COBOL **copy** verb has provided some relief, and COBOL-85 has made it easier to create generalized components. The proposed FORTRAN 90 standard (formerly 8X) also shows support for component-oriented design.

Languages such as Modula-2, Ada, and C++ show great promise. Modula-2 supports a capability for designing abstract data types but does not directly support generalized components. Ada provides excellent support for abstract data types and for generalized components in statically bound systems. Smalltalk and C++ support abstract data types but are somewhat weaker than Ada in their typing model. However, both Smalltalk and C++ seem to be superior for human interactive systems such as windowing and graphics, while Ada is probably better for environments that have a high requirement for reliability. Borland's most recent version of Turbo Pascal is close to being suitable for component-oriented software engineering, and future editions will probably be better.

Component-oriented software engineering is beginning to find champions within large organizations. Westinghouse has a corporate "reuse" newsletter devoted to designing with components, Hewlett-Packard has a software reuse committee, and the Department of Defense has made great strides toward using engineered components for a wide range of applications in Ada. New companies with engineered software components in C++ are emerging. This looks to be a major industry in the coming years. Only Ada is completely devoid of implementation dependencies and language dialects, so it is easy to develop components that are adaptable to hundreds of platforms without modifying the code—a distinct economic benefit. C++ compilers are pretty good, but there is no guarantee that good intentions will not be snookered by some weird compiler "feature."

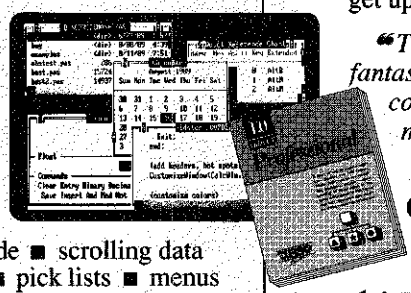
At this stage of the software industry, I see component-oriented software engineering as developing most effectively in the C++ and Ada sectors of the marketplace. Now that Ada is finding greater support outside the Department of Defense (e.g., Xerox, GE, Shell, Boeing CS), organizations such as EVB Software Engineering, Wizard Software, and the Numerical Algorithms Group will ultimately be important players in non-Defense industries. If the effort by Jerome Garfunkel at Micro Focus to establish an object-oriented COBOL is successful, we may someday see component-oriented software engineering in the MIS world as well. PJ

Save "Man-Years of Effort" with Turbo 5.5

Don't Start from Scratch with Object-Oriented Pascal

Object Professional is a huge library of over 200 object types and 2000 methods that will multiply your productivity. Window object types let you use overlapping and resizeable windows. The windows include ■ scrolling data entry screens ■ pick lists ■ menus ■ file selection ■ printed forms ■ help capability and more.

Build your programs using proven data object types like stacks, linked lists, virtual arrays, and more. System-oriented routines provide swappable TSRs in only 6K of RAM, EMS management, and much more.



Object Professional includes clear, comprehensive documentation, on-line help, full source code, technical support, and hot demo programs. Pay NO royalties. You'll get up to speed fast with OOP!

"The range of objects is fantastic. Object Professional could literally save you man-years of effort."

Jeff Duntemann

Object Professional 1.0, only $150.

A Multi-User B-Tree Toolkit

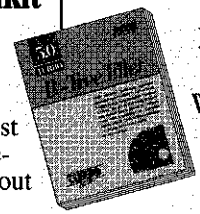
Write powerful network compatible databases faster and easier using **B-Tree Filer 5.0**. You'll have the fastest, safest, most flexible databases – no rigid structure, no TSR hassles, no running out of files. And they're compatible with Novell, 3Com, MS-NET, and others.

You get ■ Fixed and variable length records ■ Two billion records per database ■ Up to 100 indexes per index file ■ Fail-safe mode with journaling ■ Units for sorting, browsing, reindexing, and network control.

B-Tree Filer includes full source code, documentation, technical support, and you pay NO royalties.

"B-Tree Filer... a well rounded, feature-rich approach to B-Tree databases."

Computer Language, 1/90



B-Tree Filer 5.0, only $125. (single user)
With network support, $175.

Call toll-free to order.
1-800-333-4160

Satisfaction guaranteed or your money back within 30 days. Add $5 per order for shipping in U.S. and Canada. Inquire about other shipping charges. OPro requires Turbo 5.5. BTF requires Turbo 4.0, 5.0, 5.5, or QuickPascal.

TURBO Power

8AM - 5PM PST Monday through Friday, USA & Canada.
For more information call (408) 438-8608. Fax: (408) 438-8610.
TurboPower Software PO Box 66747 Scotts Valley, CA 95067-0747

Circle 25 on Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Optimization strategies

Checking the details to get a jump in performance

By John Otken

A few years ago language vendors predicted their compilers would soon generate code that would be nearly as efficient as assembly language code. They were wrong. C is a great programming language. It's complicated enough to scare away beginners, fun to program, and efficient. Of course, assembly language is more complicated, more fun, and more efficient. With due respect to Watcom, Microsoft C is about as good as 80x86 C compilers get. Bill Gates spends more dough on his C compiler than anyone else, yet it doesn't touch assembly language.

How much performance can you expect from an assembly language program? Comparing the linkers available from Borland, Microsoft, and SLR Systems offers a real-world estimate. Microsoft Link is a C program. Borland's Turbo Link is assembly language. SLR Systems's OPTLINK is a highly optimized assembly language program. Turbo Link is about twice as fast as Microsoft Link, and OPTLINK is four times faster than Microsoft.

A simplistic analysis indicates that assembly language programs are two to four times faster than C. What's more, Turbo Link and OPTLINK are significantly smaller than Microsoft Link. In addition to being smaller and faster, OPTLINK makes effective use of assembly language to handle much larger programs than Microsoft Link does. As program size increases, OPTLINK's performance can soar to ten times that of Microsoft.

CPU ARCHITECTURES

The four major CPUs running MS-DOS are the 8088, 286, 386, and 486. Because of architectural differences, there is no single way to optimize code for all four. A programmer should understand the processors in order to know when and how to optimize code.

The 8088 is crippled by four-clock memory access and a narrow eight-bit bus. Because memory bandwidth is so low, the 8088 is capable of executing most instructions faster than it reads them from memory. This leaves the prefetch queue empty most of the time. The easy way to estimate instruction times on the 8088 is to multiply the number of memory cycles required to execute the instruction by four. Do not forget to include memory access in your calculation. To optimize code for 8088, minimize memory cycles.

The 286, with its 16-bit bus, is much faster than the 8088. It reads and writes memory more quickly and executes instructions and accesses memory in fewer clock ticks.

The 32-bit bus in the 386 enables it to read instructions faster than the processor can execute them. It fetches four instruction bytes in only two clock cycles, enough for one or two common instructions. Since the fastest instructions execute in two or three clocks, the CPU cannot keep up with the instruction flow and the prefetch queue fills.

The 486 takes advantage of an on-chip cache, RISC-like performance, and higher memory bandwidth to improve on the 386. The 486 executes many instructions in one clock tick. It can read sixteen bytes of memory in only five clock ticks while the 386 requires eight.

CODE AND DATA ALIGNMENT

Code and data alignment is a sure way to improve performance on the 286, 386, or 486. Access to a word in memory is faster if the word is on an even address, and double-word access is faster if the address is a multiple of four. Access to an aligned address is faster because one less memory cycle is needed.

Code alignment affects performance at procedure entry, loop, and jump points because the CPU loses time waiting on memory when the prefetch queue is empty. To minimize lost time, the CPU must read multi-byte instructions in as few memory cycles as possible.

Assembly language has excellent alignment control. If an entry, loop, or

John Otken is president of Soft Advances where he hacks assembly language into debuggers and other software for embedded systems. You can leave him mail on his BBS: (213) 559-1449.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

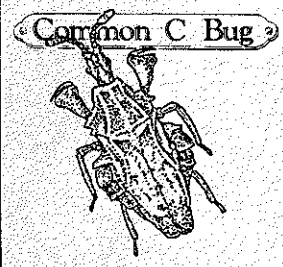
PJ ISSUE SPECIAL!

SAVE 20% ea.

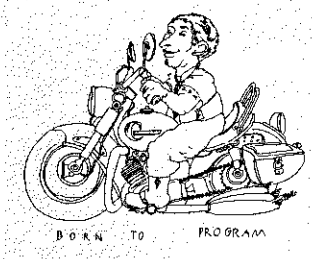
Multi-color imprints on the highest quality 100% cotton T-shirts.

SIZES: S, M, L, XL, XXL

$14.95 each — includes U.S. shipping



COMMON C BUG



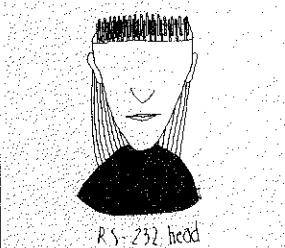
BORN TO PROGRAM

ALL T-SHIRTS NOW ONLY $11.95 EACH!

PJ T-shirts make great holiday presents for your friends — or for yourself. Order as many as you want, but order today to avoid the rush.

Free U.S. shipping. Add $2.50 shipping per shirt to Canada or Mexico, $5.00 shipping per shirt to any other foreign destination. Offer expires 12-31-90.

PARALLEL PORTS

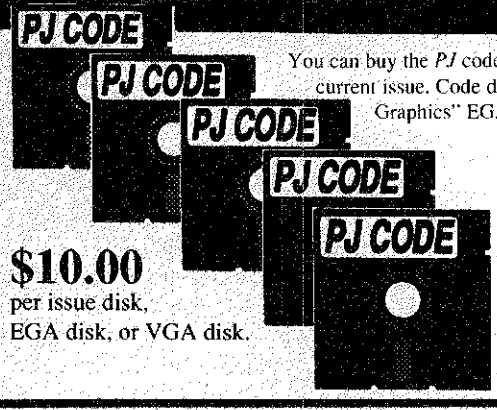


RS-232 HEAD

BUGS RULE 4 color imprint on a white shirt

All mail orders must be accompanied by check or Money Order -- PREPAID ONLY.

Use the order form on the card opposite.
Order by phone with your VISA/MC.
CALL TOLL FREE
800/234-0386 ext 53



$10.00 per issue disk, EGA disk, or VGA disk.

You can buy the *PJ* code listings from volume 5.5 to the current issue. Code disks for the Michael Abrash "On Graphics" EGA/VGA series are also available.

Diskettes are available for the following issues:

EGA	6.1	7.1	8.1
	6.2	7.2	8.2
VGA	6.3	7.3	8.3
	6.4	7.4	8.4
5.5	6.5	7.5	8.5
5.6	6.6	7.6	8.6

360K MS-DOS disks standard; other formats are available.

$7.00 each PREPAID ONLY

A few of the Back Issues currently available

Vol. 7.1: GRAPHICS: High-resolution VGA color modes; graphics processors; Turbo Pascal 4.0 and 5.0 compiled EXE code; Pascal overlay managers; JPI's TopSpeed Modula-2; faster QuickBASIC code; Microsoft's CodeView debugger; COBOL-85 ANSI standard.

Vol. 7.2: OPERATING SYSTEMS: Undocumented DOS features; fast 80386 block moves under DOS; OS/2 dynamic link libraries; OS/2 multitasking; debugging utility for PM and Windows; EGA and VGA animation; disk formatting in Pascal; reviews of windowing and graphics tools, disassemblers, and *The 80386 Book*.

Vol. 7.4: COMMUNICATIONS: A simple terminal program in PM; communications and protocols; database normalization theory and practice; Turbo Pascal 5.5 development tools; Lint utilities for the PC; a millisecond timer for BASIC; reviews of RamNet, DOS/16M, PCX *Technical Reference Manual* and Programmer's Toolkit.

Vol. 7.5: OBJECT-ORIENTED PROGRAMMING: Class libraries in C++; from C to objects; improved C++ code generation; Hal Hardenbergh on the RISC/CISC schism; world scaling in Turbo Pascal's BGI graphics primitives; numeric extensions to C; reviews of 386 development tools, mini-Case tools, and Robert Sedgewick's book *Algorithms*.

Vol. 7.6: USER INTERFACES: Windows terminal program; the CURSES C library; two video displays with Turbo Pascal; VGA and EGA split screens; memory models and static variables for assembly language; object-oriented programming to create a file viewer; large memory spaces from DOS on the 80386; QuickBASIC's string-searching algorithm; reviews of user-interface and mini-Case tools; the *System BIOS for IBM PC, XT, AT*.

Vol. 8.1: NETWORKING: Network File System; an overview of NetBIOS on the PC, from *C Programmer's Guide to NetBIOS* by W. David Schwaderer; fast circle drawing on the VGA; dynamic data exchange in Windows; DESQview API features; computing time in C; a menu system with Turbo Pascal 5.5.

Vol. 8.2: GRAPHICS: Faster circles for the VGA; the TIGA interface from TI; the EGA palette; 800x600x16 Super-VGA mode programming; dynamic data exchange for communications; manipulating large data; fast file I/O in QuickBASIC; industry news; 1989 article index.

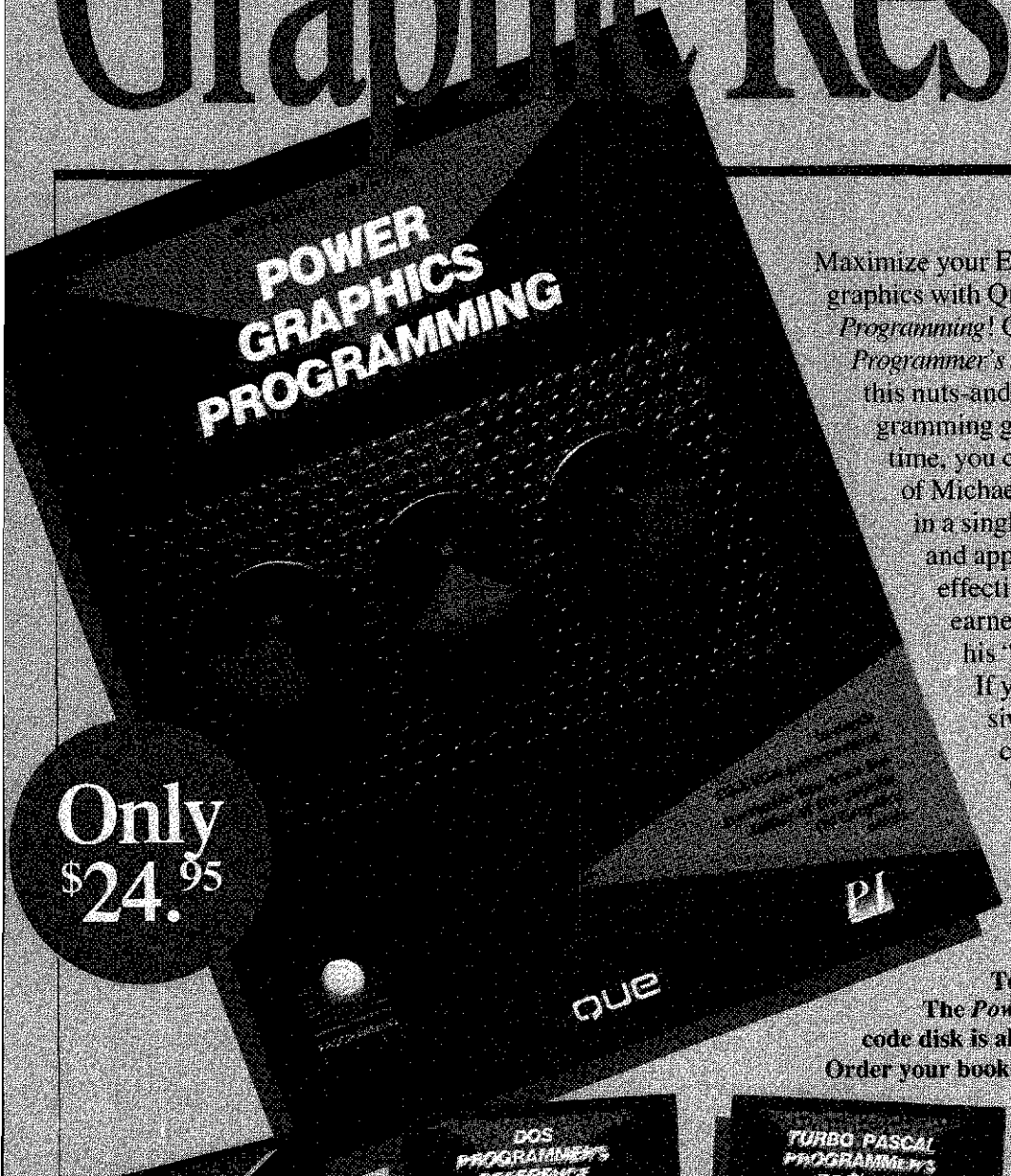
Vol. 8.3: OPERATING ENVIRONMENTS: Undocumented DOS shell; graphics programming in Windows; mousing around: messages in PM; accessing high memory; ASM:advanced segmentation; ellipses on the VGA; up OOP alley in Pascal.

Vol. 8.4: PROGRAMMING IN THE PROTECTED MODE: Faster VGA ellipses and more; running unprotected virtual-8086 mode; converting to 386 protected mode; OS/2 2.0 memory models for the 386; Fortran for the 386; using C++ features in C; compensating for system clock idiosyncracies; ASM: errors and stream I/O; Cobol-85 nested; deliberately breaking the rules.

10% discount on any order of 10 or more T-Shirts, Disks or Back Issues.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Graphic Results!

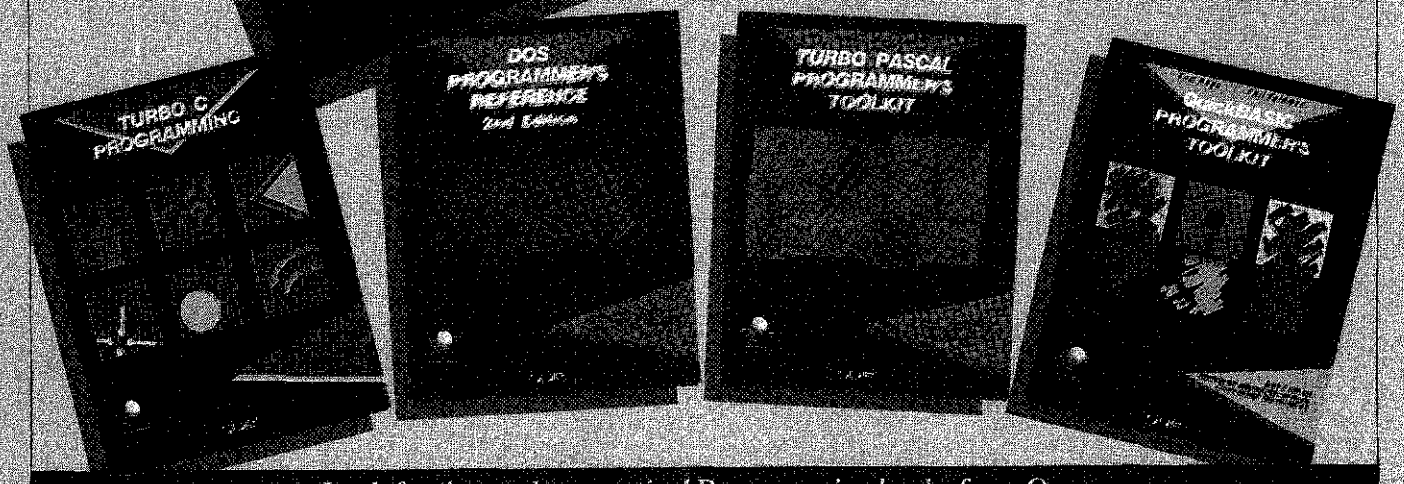


Only $24.95

Maximize your EGA/VGA programming graphics with Que's *Power Graphics Programming!* Que has teamed up with *Programmer's Journal* to bring you this nuts-and-bolts graphics programming guide. Now, for the first time, you can have the expertise of Michael Abrash consolidated in a single volume. His insights and approach to efficient, effective programming have earned him high acclaim for his "On Graphics" series. If you are seeking that elusive graphic edge, you can't be without *Power Graphics Programming!*

Order yours directly from *Programmer's Journal*

To order call 800-234-0386. The *Power Graphics Programming* code disk is also available for $14.95. Order your book and code disk today!



Look for these other practical Programming books from Que

Available in better bookstores and computer stores everywhere. Or call 800-428-5331, ext. PJ01.

que®

Circle 296 on Reader Service Card

PRODUCED 2004 BY OMZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

jump point executes infrequently, there is no need to waste bytes on alignment. The performance improvement is negligible. Aligning a procedure that executes 100 times on a 10 MHz 286 saves a meager thirty microseconds. You have to execute the program thirty thousand times to save a second!

A simple assembly language directive for alignment control is **even**. If the location pointer is odd, **even** generates a **nop** (90h); otherwise, **even** has no effect on the code. Use **even** to word-align code for 286s. The following procedure aligns a loop on a word boundary:

```
;;      linear search
;
;       entry   AX      word to search for
;               CX      number of entries (0)
;               DX      table entry byte count
;               DS:SI   table pointer
;       exit    SI,CX   updated
;               Zf      if found
;
linear_search proc
        sub     si,dx
        even            ; force loop alignment
lns1:   add     si,dx
        cmp     ax,[si]
        loopne  lns1
        ret
linear_search endp
```

The biggest performance gain comes from loop alignment because loops are easy to identify and programs spend significant time in them. In the example above, the procedure entry point is less important than the three-instruction loop because it doesn't execute as much. Adding a **nop** to increase performance is a programming paradox, but the savings in the loop more than compensate for the expense of the **nop**.

Even alignment helps the 286 and 386 alike; however, the 386 performs better still with double-word alignment. When the 386 jumps to a double-word address, it gulps down four instruction bytes on the first read. That is usually enough for the current instruction and a byte or two for the prefetch queue.

The **align** directive provides general-purpose alignment control. It aligns the location pointer to any boundary. For double-word alignment use:

```
align 4   ; double word alignment
```

OPTASM's **align** accepts three arguments. If a second argument appears, it specifies a fill value. By default, OPTASM fills space with **nop**s. It uses **xchg AX,AX** for a single-byte **nop** and **mov AX,AX** for a double-byte **nop**. Under OPTASM, the third **align** argument is an offset. In a double-word alignment, you can offset to any of the four bytes within the double word. The default offset is zero.

OPTIMIZING OFFSETS

Some clever optimizations are possible with **align** offsets. Consider an important procedure starting with a near **call** on a double-word boundary plus one. The three-byte **call** still fits in a double word and the instruction after the **call** aligns to a double word. On entry, the 386 reads the **call** in one memory cycle, executes the subroutine, and returns to perfectly aligned instructions!

```
                align  4,,1
        ;;      strcpy
        ;
        ;       entry  DS:SI  string pointer
        ;              ES:DI  destination ptr
        ;       exit   DI,SI  updated
        ;       uses   AX,CX
        ;       note   this is sample code;
        ;              it is NOT optimized.
        ;
        strcpy  proc
E8 rr rr        call   strlen
F3/ A4          rep    movsb
A4              movsb
C3              ret
        strcpy  endp
```

The alignment offset makes **strcpy** fit perfectly into memory, as shown in Figure 1 below:

FIGURE 1

??	E8	rr	rr
F3	A4	A4	C3

FIGURE 2

E8	rr	rr	F3
A4	A4	C3	??

Were **strcpy** double-word aligned without an offset, as shown in Figure 2, the 386 would return and read the **F3** and **A4** bytes in separate memory cycles before it could execute the **rep movsb**.

MASM and TASM lack OPTASM's offset feature, but they can generate an offset manually. This technique is not space-efficient, but it gets the job done. For example, **align 4,,1** is similar to:

```
ALIGN 4         ; double word align
NOP             ; fill single byte offset
```

When aligning procedure entry points, do not put **align** or **even** after the **proc** directive or else it will generate **nop**s after the entry point. You want the **nop**s before the entry point (where they will never execute).

```
;;      strcpy
;
strcpy  proc
        even    ; WRONG WRONG WRONG!
cpy1:   lodsb
        stosb
        cmp     al,0
        je      cpy1
        ret
strcpy  end
```

Make sure that **align** does not exceed the alignment of a segment. If a segment is word-aligned, double-word alignment is impossible. Unfortunately, simplified segmentation will word-align the code segment. Double-word alignment requires full segment directives, as shown below. (See this column in issue 8.4 for more on full segmentation.)

```
_TEXT segment dword public 'CODE'
_TEXT ends
```

Because the 486 cache fills in chunks of 16 bytes, critical procedures can benefit from paragraph alignment on the 486. This is a radical optimization for perfectionists who can't sleep if their programs waste a clock tick. Loops do not benefit from this trick because they quickly become cache-resident regardless of their alignment.

DATA ALIGNMENT

Data alignment is another method to improve performance. While a programmer may align only a small percentage of code, all 16-bit and 32-bit data objects should be aligned to a word boundary. Maintaining data alignment is easy. When you define data objects, specify 32-bit objects first, then 16-bit, then eight. This order ensures that smaller objects do not interfere with the alignment of larger ones.

As with code segments, simplified segment directives word-align data segments. For optimum data alignment on

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

386 and 486 systems, you must use full segment specifications, such as:

```
_DATA segment dword public 'DATA'
_DATA end
```

The data structure most sensitive to alignment is the stack. If the stack pointer is odd, program performance will stink. Simplified segmentation protects you from this error because it paragraph-aligns the stack.

STRUCTURE ALIGNMENT

For fast structure indexing, order structure members from large to small. This ordering ensures proper alignment for members when the structure begins on a double-word boundary. For better program performance, the first structure member should be the one that is accessed most often by the program. Be sure to arrange the member positions to keep alignment. If there are no appropriate members, fill the space with a nameless member.

```
type_record struc
  type_id    db  ?  ; (important member)
             db  ?  ; (filler)
  type_count dw  ?
  type_size  dw  ?
type_record ends
```

Put **dup**ped members at the end of the structure.

```
receiver struc
  pointer dw ?
  buffer  db 512 dup(?)
receiver ends
```

EFFECTIVE ADDRESSES

The last two tips for optimizing structure declarations make more sense if you understand the 80x86 effective address mechanism. When an 80x86 CPU executes an instruction with a memory access, it can use any of nine different algorithms to calculate the address.

Eight of the algorithms add one or two index registers and an optional offset. If the offset is present, it is either eight or 16 bits. The ninth algorithm is a simple 16-bit direct address; no index registers are involved.

A byte after the opcode selects the address algorithm and the size of the offset. This value is the *mod r/m* byte. The *mod* field contains the upper two bits of this value and specifies the size of a displacement as zero, one, or two bytes. The *r/m* field sits in the lower three bits of *mod r/m*; its function is to specify one of eight possible indexing combinations.

When the assembler builds a *mod r/m* byte, it uses the smallest possible offset. Assuming an index register points to a structure in memory, the offset of the first structure member is zero. With no offset, the instruction is shorter and executes faster. Thus, the first structure position has a slight performance edge over the other members.

Members in the next 126 (0x7E) bytes of the structure require a single byte offset. When the CPU reads this byte from the instruction stream, it sign-extends to 16 bits before adding it to an index register. Members at offsets 128 (0x80) and greater require a full 16-bit offset in the instruction.

If you place buffers at the front of a structure, instructions that access other data objects are larger than necessary. In the sample below, notice the two-byte offset (0200) in the **mov** instruction.

```
                receiver struc
0200[ ?? ]      buffer  db 512 dup(?) ; WRONG!
????            pointer dw ?
                receiver ends

8B 9C 0200  mov  bx,pointer[si]
```

By changing the order of the members, the same instruction in the following fragment becomes two bytes shorter. This saves two memory cycles on the 8088 and one on the 286. It may or may not help a 386.

```
                receiver struc
????            pointer dw ?
0200[ ?? ]      buffer  db 512 dup(?)
                receiver ends

8B 1C       mov  bx,pointer[si]
```

Of course, member alignment is only as useful as the alignment of the structure itself. If the structure is word aligned in memory, don't waste time and space on double-word member alignment.

Ordering data objects from large to small is just a guideline. Any order that maintains alignment will work as well, but large to small is easier to explain.

REGISTER CONVENTIONS

The 80x86 has seven "general purpose" sixteen-bit registers that hold either numeric or address objects. Three of these registers—AX, CX, and DX—are more suitable for numeric objects. You can keep an address in them, but you cannot use the register as a memory index, except for the 386 and the 486.

The four index registers—BP, BX, DI, and SI—are useful for address objects. The CPU can access memory based on a pointer in one of these registers. If none of the three numeric registers are available in a procedure, an index register can moonlight as a numeric register.

Most 80x86 instructions that access registers have forms which designate a specific register. From these instructions, one can derive a convention for register usage. It is very important to follow these register conventions as much as possible, even in the absence of any special instructions.

When register conventions between MS-DOS and the CPU conflict, ignore the MS-DOS conventions. It is easy to create an interface to MS-DOS using the native register conventions. It is impractical and inefficient to use MS-DOS register conventions in almost any assembly language program.

AX is the most valuable numeric register. Eleven instructions—**adc**, **add**, **and**, **cmp**, **mov**, **or**, **sbb**, **sub**, **test**, **xchg**, and **xor**—have special opcodes for AX and AL that are smaller and faster. In the sample below, notice the different lengths of the two **mov**s.

```
A1 0000     mov     ax,loop_count
8B 0E 0000  mov     cx,loop_count
```

BX has one special purpose instruction, **xlat**. I doubt that you will ever use this instruction, so consider BX a general-purpose register. BX is a good index register and it works well as a pointer to complicated structures. BX is able to double-index with either SI or DI. BX is more useful as a numeric register than BP, DI, or SI because the CPU can access the upper and lower bytes of BX directly as BH and BL.

CX is the looping register. Four instructions designate operation on CX: **jcxz**, **loop**, **loope**, and **loopne**. Of those, **jcxz** and **loop** are very useful. The lower eight bits in CL are used to hold the bit count for the five or so shift and rotate instructions.

Even in cases where one of the special instructions does not apply, CX is the preferred choice for counts and for size objects. For example, **strlen** returns a string length in CX and **malloc** accepts a storage byte count in CX. CX is a numeric register. The CPU cannot

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

access memory with CX, though a program may temporarily store an address object in CX.

DX, a numeric register, is most often a general-purpose register. Seven instructions specify DX: **cwd**, **div**, **idiv**, **imul**, **in**, **mul**, and **oit**. The multiply and divide instructions pair DX with AX as a 32-bit register. Instructions **in** and **out** keep an I/O port number in DX. Keep in mind that the CPU cannot access memory with DX. Programs sometimes store address objects in DX for comparisons. For example, DX might point to the end of a buffer. A program detects an end-of-buffer by comparing DX with an index register such as SI.

DI is the input pointer for **cmps** and the output pointer for **movs** and **stos**. Despite **cmps**, DI is the preferred output index register. After SI, DI is the next-best memory pointer. Watch DI's default segment register. Depending on the instruction, DI uses three segment registers:

```
stosb           ; ES:
mov  al,[di]    ; DS:
mov  al,[di+bp] ; SS:
```

DI is the index register of choice for the **les** instruction. High-level language code generators take exception to this convention, one of many reasons their code executes so slowly. In many cases, using BX with **les** is a mark of poor assembly language.

Only two instructions designate SI for a specific function: **lods** and **movs** both read memory from a pointer in SI. The subtle influence of these instructions is enough to make SI the most useful address register in assembly language programs.

The list of when not to use SI is shorter than when to use it. Use BP to address the stack segment. Point BX at a complex structure base, leaving SI free to access buffers within the structure. Use DI for an output pointer.

The fragment below shows how to use BX and SI to access a structure. BX points to the structure base and SI holds an index into the buffer.

```
serial_io struct
  sio_in  dw ?          ; BX points at struct base
  sio_out dw ?
  sio_buf db 64 dup(?); SI points into buffer
serial_io ends

lds  bx,serial_io_ptr ; DS:BX points at structure
mov  si,sio_in[bx]    ;  and SI is buffer index
mov  al,sio_buf[bx+si]; grab buffer character
inc  si               ; update buffer index
mov  sio_in[bx],si    ;  ignores overflow
```

BP has no special instructions, but it has a critical function in most programs. When an instruction indexes with BP (either alone or with DI or SI), then the default segment is SS. All high-level language code generators rely on this function to access local variables and parameters on the stack.

Because all high-level languages use BP to access local variables and parameters, you should avoid using BP as a general-purpose register. Don't ever pass or return parameters in BP. If you use BP in a procedure, always make sure that you preserve its contents before returning.

CS points to the segment in memory where the CPU is executing instructions. Avoid using CS to access data. It is a bad programming practice and almost never necessary.

MORE REGISTERS

DS points to the primary data segment. Assembly language programs access most of their data with this register. High-level languages and the Microsoft memory models fix DS to **dgroup**. To access a segment other than **dgroup**, they use ES. Assembly language programs frequently change DS to access large data spaces because DS access is more efficient than ES.

For upward compatibility with protected mode, never load an invalid segment into DS (or any segment register). The only exception to this rule is zero. You can load a NULL pointer into DS, but you cannot access memory with the pointer. Never add, subtract, or multiply a segment value. In the protected mode, segments do not correspond to physical memory addresses.

The only instruction that specifies DS is **lds**. For ES, the instruction is **les**. Use these instructions to load 32-bit pointers into DS:SI and ES:DI.

ES is a secondary data segment register. Use ES to access large data spaces when DS is busy. ES:DI forms the default segment:offset pair for **cmps**, **movs**, and **stos**.

SS points to the stack segment. Because most programs group the stack segment into **dgroup**, SS is a good way to access **dgroup** variables when DS points to another segment.

The 386 has an **lss** instruction similar to **lds** and **les**, but it is not available on the 286. Since most programs set and forget the stack pointer, the instruction isn't very useful.

AN ALPHABETICAL GUIDE

The following guide gives some tips on how to optimize code with the various instructions. They are listed in alphabetical order, for handy reference.

The instruction **aad** multiplies AH by ten and adds the product to AL. It leaves a zero in AH. You can use **aad** to pack decimal digits into a binary number, as follows:

```
sub  ax,'00' ; convert ASCII digits to decimal
aad          ; convert decimal digits to binary
```

The next instruction, **aam**, divides AH by ten, stores the result in AH, and the remainder in AL. Use **aam** to convert a binary number less than 100 to ASCII characters:

```
mov  al,ah   ; convert binary # <100 in
aam          ;  AL to two decimal digits
add  ax,'00' ; make decimal digits ASCII
```

I believe everyone knows how to **add** instead of multiplying by powers of two. Less obvious is **add**ing up other products.

```
add  ax,ax  ; multiply AX by 10
mov  dx,ax
add  ax,ax
add  ax,dx
```

Do not use **sal** or **shl** instead of **add**—**add** is easier to read. Only use **sal** or **shl** when you mean "shift."

When masking the bits in AL with **and**, you can zero AH for only an extra byte.

```
and  al,7Fh  ; 4 bytes
mov  ah,0

and  ax,7Fh  ; 3 bytes
```

A far **call** to an address in the same segment can be optimized into a **push** and a near **call**. The replacement is one byte shorter and much faster in protected mode. OPTASM does this optimization automatically.

```
call  far_proc           ; 5 bytes

push  cs                 ; 4 bytes
call  near ptr far_proc
```

The near **call** instruction is three bytes long. A program can trade size for speed by calling common code sequences that are longer than three bytes.

The instruction **cbw** offers a tempting way to zero AH when you know AL

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

is less than 80h. I've seen this bite someone who ported the code to OS/2 and found the command tail byte count more than 80h.

The carry flag can be cleared by the one-byte instruction **clc**. Because of **clc** (and **stc**), the carry flag is the best way to return a boolean value from a procedure. To set the zero flag, compare a register to itself.

```
cmp  ax,ax  ; set zero flag
```

Comparing a register to zero is equivalent to **or**ing the register to itself. See **or** below for an example.

You can zero DX efficiently or compute the absolute value of AX with the instruction **cwd**.

```
xor  ax,ax  ; zero AX and DX
cwd
cwd         ; absolute value
xor  ax,dx  ;  of AX (DX bashed)
sub  ax,dx
```

Intel should have called the **daa** instruction **cth** for "convert to hex." Ever since the 8080, assembly language programmers have used the **daa** instruction in order to convert four bits into an ASCII character.

```
and al,0Fh  ; convert the low 4 bits of
add al,90h  ;  AL to an ASCII character
daa
adc al,40h
daa
```

Use one or two **dec** instructions in 16-bit registers instead of **sub**tract and save a byte or two. It is not a direct replacement because **dec** does not affect the carry flag, while **sub** does.

```
sub  ax,2  ; 3 bytes

dec  ax    ; 2 bytes (carry flag
dec  ax    ;  is not affected)
```

When you decrement the low byte of a register, you can squeeze a byte out of the instruction by using a 16-bit decrement. It bashes the upper byte of the register and obscures the code, so use this sparingly.

```
dec  ax    ; DEC AL optimized
```

A stack frame for local variables can be created with **enter**. It is the same as the standard three-instruction sequence, except that it does not affect the flags. This instruction is not available on the 8088 and 8086.

```
enter  8,0    ; 4 bytes

push   bp     ; 6 bytes
mov    bp,sp
sub    sp,8
```

The 286 has a useful three-operand version of **imul**. It multiplies any memory or register object by an immediate value and stores the result in any word register. Since the three-operand **imul** discards the upper half of a 32-bit product, you can use **imul** for signed or unsigned multiplies.

Short versions of port **in**put and **out**put instructions are available for the first 256 ports.

```
mov dx,20h  ; 5 bytes
in  al,dx

in  al,20h  ; 2 bytes
```

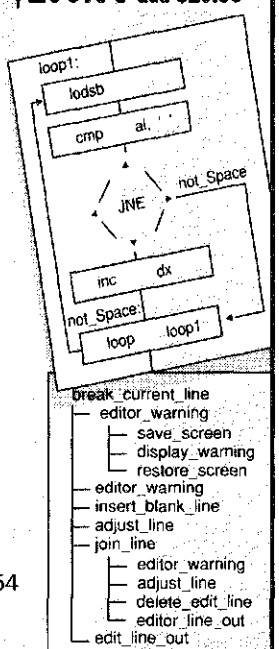

ASMFLOW PROFESSIONAL

YOU CAN STEP BACK AND TAKE A LOOK AT YOUR CODE

$199.95 S/H $5.00 Outside U.S. add $10.00

FLOW CHART AND ANALYZE YOUR ASSEMBLY LANGUAGE SOURCE CODE

- Flow Charts
- Tree Diagrams
- Stack Sizing
- Register Analysis
- CPU Timing Analysis
- Procedural X-Reference
- 8088/87 to 80486
- Context-Sensitive Help
- MASM 5.1 Compatible
- Macro Expansion
- 8085, Z-80 and 8051 Versions



QUANTASM CORPORATION
19855 Stevens Creek Blvd, Suite 154
Cupertino, CA 95014
(408) 244-6826 – Free Demo

VISA •MC 30 – DAY MONEY BACK GUARANTEE

Circle 252 on Reader Service Card

We Have Ways To Increase Your Productivity

TE Developer's Kit ($139)

Why reinvent the wheel? TE Developer's Kit provides the most cost effective way of incorporating text editing features into your application. The kit includes complete source code of the editor and the executable file. Supports Microsoft and Borland 'C' compilers.

The Features Include: multiple files/windows, word-wrap and printing, edits files larger than available memory, reconfigurable keyboard commands and screen colors, keystroke macros, undo, column block commands, search/replace, and autosave. **The kit also includes TER small editor routine. TER is a subset of TE. Your application calls this routine by specifying window location and size, maximum file size, and an input buffer or file.**

WinMem ($95): Windows 3 Memory Manager

WinMem provides an unlimited amount of virtual memory even in Windows' Real and Standard modes. **The device independence is not enough. WinMem offers you CPU and memory resource independence.** It provide an unlimited number of global memory handles, and reduces the overhead per block to 4 bytes (from 20 bytes of Windows). WinMem will trap any attempt to free an illegal memory block. This feature alone can save you a countless number of hours of debugging memory problems. Includes the complete source code.

Team ($119): The Programmer's Text Editor

An advanced text editor for a not so advanced price. **Features:** edits unlimited size files, unlimited levels of undo/redo, multiple zoomable windows, regular expression search/replace, editable keystroke macros, 'C' like macro language, task switching, customizable context sensitive help, customizable keyboard commands, menu with accelerator keys, frees entire memory for DOS commands, 132 column screen mode, and much more.

Sub Systems, Inc. 1-800-447-6819
Fax: 1-617-438-8901
159 Main Street, #8C, Stoneham, MA 02180, 1-(617)-438-8901

Circle 68 on Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

The instructions **inc** and **dec** are related. (See **dec** above.) A software interrupt is invoked by **int**. From the CPU's perspective, there is no difference between an **int** instruction and a hardware interrupt. You can simulate an **int** with the four-instruction sequence:

```
pushf
cli
push    cs
call    near ptr foo
```

The instruction **jcxz** branches if CX equals zero. It does not require a compare or test instruction, nor does it affect the flags. It is a quick way to test for a NULL pointer.

```
lds   si,pointer
mov   cx,ds
jcxz  fool        ; if NULL pointer
```

The fastest way to test for both segment and offset equal to zero is:

```
lds  si,pointer
mov  ax,ds
or   ax,si
jz   fool         ; if NULL pointer
```

Two uncommon forms of **jmp** optimize some programs. The first is jumping through a jump table or state variable. The second is a jump to an address in a register.

```
menu_choice dw 1
menu_tab dw menu1,menu2,menu3
   ...
   mov  bx,menu_choice
   add  bx,bx
   jmp  menu_tab[bx]

process_state dw default_state
   ...
   jmp   process_state    ; (jump indirectly)

   mov   cx,process_state
   jcxz  foo3             ; if no state set
   jmp   cx               ; else process state
```

The instruction **lds** loads a 32-bit pointer into DS and some general-purpose register, usually SI or BX. It is shorter and faster than separate **mov** instructions to DS and the register.

```
C5 36 0000  lds  si,pointer            ; 4 bytes

8B 36 0000  mov  si,word ptr pointer  ; 8 bytes
8E 1E 0002  mov  ds,word ptr pointer+2
```

There is no corresponding store instruction to **lds**. You have to store the offset and segment registers with separate **mov** instructions.

A register can be added or subtracted without modifying the flags by using the instruction **lea**.

```
fool: cmp   al,type_id[si]
      lea   si,[si+size type_record]
      loopne fool
```

Both **lea** and **mov** immediate can load an address into a register. Although **mov** is one byte shorter than **lea**, it's not as readable. OPTASM generates the **mov** instruction for **lea** when possible.

```
8D 36 0006  lea si,buffer          ; 4 bytes
BE 0006     mov si,offset buffer   ; 3 bytes
```

In addition, **lea** can add one or two registers and a number and store the result in another register. The only restriction is that the register combination must be a valid effective address. Be sure to comment any lines that use this trick.

```
mov ax,bx              ; 7 bytes
add ax,si              ; (ADD modifies the flags,
add ax,number          ;  LEA does not)

lea ax,[bx+si+number] ; 3 or 4 bytes
```

The instruction **leave** destroys a stack frame. It reverses the effects of **enter**. Note that **leave** is not available on the 8088 and 8086.

```
mov  sp,bp             ; 3 bytes
pop  bp

leave                  ; 1 byte
```

Otherwise identical to **lds**, **les** loads into ES instead of DS. DI is the most common register paired with ES.

A fast way to access consecutive objects in memory is with **lods**. This instruction can access either word or byte objects.

```
mov  al,[si]   ; 5 bytes
lea  si,[si+1]

lodsb          ; 1 byte
```

The instruction **lods** usually appears with the data object size in the mnemonic: **lodsb** and **lodsw**. If you want the assembler to select the object size, use a typed symbol with **lods**. Also, note that the **rep** prefix is not useful with **lods**.

```
lods  function_code[si]
```

Using **loope** and **loopne** can eliminate an instruction or two from search loops. The following example searches a list of structures in memory.

```
      sub    si,size type_record ; search type
fool: add    si,size type_record ; records
      cmp    al,type_id[si]
      loopne fool
      jne    foo9                ; if not found
```

Don't use **jcxz** to test for a failed search after **loope/loopne**. If a match occurs when CX==1, the **jcxz** branches when it should not.

MOV INSTRUCTIONS

There are about five different **mov** instructions. The 80x86 can move immediate values to any general purpose register. The 80x86 cannot move an immediate value to a segment register in one instruction. You must first move the immediate value to a general purpose register and then move the value from there to the segment register.

```
mov    ax,@data
mov    es,ax
```

While there is no immediate move to a segment register, you can move between a segment register and memory.

```
mov  es,dgroup_segment
```

Moving values between segment registers requires two instructions. You can select either a fast or small sequence.

```
mov  ax,ds  ; 4 bytes, fast (use
mov  es,ax  ;  any register)

push ds     ; 2 bytes, slow
pop  es
```

A fast instruction for moving large blocks of memory is **movs**. The following four lines implement a fast storage move. The byte count in CX can be odd or even. The best performance with this instruction occurs when the data aligns to word boundaries. AX is not modified by **movs**.

```
shr  cx,1  ; divide byte count by 2 and
rep  movsw ;  move words
adc  cx,cx ; move final byte if byte
rep  movsb ;  count odd
```

Negating a 32-bit number requires three instructions:

```
neg  dx    ; negate DX AX
neg  ax
sbb  dx,0
```

The instruction **or** is a smaller and faster substitute for comparison with zero. Assuming the stack pointer is non-zero, **or** can clear the zero flag.

```
cmp  ax,0  ; 3 bytes
or   ax,ax ; 2 bytes
or   sp,sp ; clear Zf
```

The single-byte instructions **popa** and **pusha** push or pop most of the general-purpose registers. They do not operate on the segment register. Because they are slow, avoid using these instructions unless a procedure has long lists of **push**

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

and **pop** instructions. In addition, **pusha** and **popa** are not available on the 8088 and 8086.

The instruction **push** immediate offers an interesting optimization for a list of **call** instructions. Reverse the list and translate all but the last **call** into a **push** immediate.

```
call  fee          ; 12 bytes
call  fie
call  foe
call  fum

push  offset fum   ; 12 bytes
push  offset foe
push  offset fie
call  fee
```

The two sequences are the same length, but the second version executes three fewer "jumps." Unfortunately, **push** immediate is not available on 8088 and 8086 CPUs.

FAST SEARCHING

Use **scas** as a fast search instruction for large blocks of memory, but note that its register designation of ES:DI is inconvenient. For short sections, you'll find that a simple search loop is smaller and faster than juggling registers to use the **scas** instruction.

```
fool: pushm  ds,es    ; (as you can see,
      popm   ds,es    ;  swapping DS:SI
      xchg   si,di    ;  and ES:DI is a
      repne  scasw    ;  multi-line pain
      pushm  ds,es    ;  in the rear.)
      popm   ds,es
      xchg   si,di    ; (14 bytes)
fool: cmp    ax,[si]  ; (assumes CX!=0)
      lea    ax,[si+2]
      loopne fool     ; (7 bytes)
```

Use **sub** to zero a 16-bit register faster and in one less byte than a move immediate. Also, **sub** alters the flags while **mov** does not.

The one-byte **xchg** with AX offers a tempting replacement for some move instructions. If bashing either AX or the target register does not affect your algorithm, then you may want to substitute **xchg** for **mov** and save one byte. Unless you really need to save space or align instructions, I recommend against it, however. Exchange is slower on most CPUs and it is much harder to read than a move.

```
      lodsw
      xchg  cx,ax ; (XCHG opcode is one byte)
fool: ...
      loop  fool

      lodsw
      mov   cx,ax ; (MOV opcode is two bytes)
fool: ...
      loop  fool
```

Like **sub**, the instruction **xor** can zero a 16-bit register faster and in one less byte than a move immediate can. Another difference is that **xor** alters the flags and **mov** does not.

```
mov ax,0   ; 3 bytes
xor ax,ax  ; 2 bytes
```

IN THE FUTURE

Many more issue-optimizing techniques remain for my next installment. I am also working on some very high performance I/O routines. The windowing code from the last column was too long for available space in *PJ*. This, too, awaits a future issue. PJ

Tired of the limitations of MS-DOS?
Thinking about switching to another OS?
Well DON'T, now there is:

PRO-CLI v4.1

CLI is a tiny, intelligent front & back end processor that enhances the MS-DOS system kernel. CLI provides many features of advanced OS's, like VMS & UNIX, to MS-DOS without the tremendous costs and additional training.

CLI-Types: ALIASs, LOGICALs, SYMBOLs, LEXICAL-FUNCTIONs and STREAMs are available to the command line, batch programs, & all existing application programs. CLI is 100% MS-DOS compatible, and NO changes are required in any software to reference any CLI-Types.

Some of the features offered by PRO-CLI include:

- batch programs with numeric & string processing
- facilities for source code and project management
- transparent task-to-task communications
- file & device management from the command line
- command line editing, history, and recall
- flexible system installations & re-configurations
- user configurable PRO-CLI resources

For the first time, designers, developers, and managers can coordinate software development & production efforts without the major restrictions of MS-DOS.

Call or Write for Additional Information!

Trademarks: VMS of D.E.C. MS-DOS of Microsoft UNIX of AT&T

Alphax Systems, Inc.
1005 Sussex Blvd.
Broomall, PA 19008
(215) 543-2658

Price: $ 149
S/H: $ 6

Major Credit Cards Accepted

Circle 345 on Reader Service Card

Total control with *LMI FORTH*™

For Programming Professionals: an expanding family of compatible, high-performance compilers for microcomputers

For Development:
Interactive Forth-83 Interpreter/Compilers for MS-DOS, OS/2, and the 80386

- 16-bit and 32-bit implementations
- Full screen editor and assembler
- Uses standard operating system files
- 500 page manual written in plain English
- *Support for graphics, floating point, native code generation*

For Applications: Forth-83 Metacompiler

- Unique table-driven multi-pass Forth compiler
- Compiles compact ROMable or disk-based applications
- Excellent error handling
- Produces headerless code, compiles from intermediate states, and performs conditional compilation
- Cross-compiles to 8080, Z-80, 8086, 68000, 6502, 8051, 8096, 1802, 6303, 6809, 68HC11,34010, V25, RTX-2000
- No license fee or royalty for compiled applications

Laboratory Microsystems Incorporated
Post Office Box 10430, Marina del Rey, CA 90295
Phone Credit Card Orders to: (213) 306-7412
FAX: (213) 301-0761

Circle 63 on Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

PJ PRESSROOM

OS/2 MULTISCOPE DEBUGGER

Logitech is now shipping its MultiScope Debugger, a powerful multi-language debugger designed specifically for OS/2 development. Compatible with any language which generates standard CodeView OS/2 debugging information, MultiScope works in both OS/2 text and Presentation Manager mode.

MultiScope offers both runtime and post-mortem debugging options. The runtime debugger controls program execution with a complete set of execution control features and stepping commands. The post-mortem debugger, an exclusive Logitech feature, captures the program execution state of an application running outside the debugger at the time of the crash. The resulting dump file can easily be examined to determine the state of the program when it crashed. MultiScope also includes support for debugging threads, dynamic link libraries, and child processes.

In addition to displaying data line-by-line, MultiScope shows complex data structures as easily-understood graphical structures. Multiple overlapping windows can display as many as 13 different program views simultaneously. Developers can create custom configurations of source, assembly, data, register, memory, module, breakpoint, watchpoint, graphical data, threads, call chain, log, and output views. The MultiScope Debugger is priced at $299.

Logitech, Inc.
6505 Kaiser Drive
Fremont, CA 94555
(415) 795-8500

UNIX-COMPATIBLE COHERENT

Mark Williams Company has introduced Coherent, a UNIX-compatible operating system for IBM AT-compatible 286 and 386 machines. An independently-developed version of UNIX, Coherent is a multi-user, multitasking operating system that is smaller, faster, and less expensive than UNIX.

The Coherent kernel is 64K; the entire system requires only 10 megabytes of disk space. The system comes with Lex, Yacc, a complete C compiler, a Bourne shell, and the full set of nearly 200 UNIX commands (with text processing, program development, and administrative and maintenance commands). Also included is a version of UUCP for remote communications, a MicroEmacs editor, Kermit, and utilities for accessing DOS floppy diskettes and hard disk partitions.

Aside from the bundled software, a number of public-domain UNIX programs have been ported to Coherent. Several commercial programs have also been ported, such as RDB, a relational database manager also available on SUNs, VAXs, and 386 UNIX systems.

Coherent runs in a minimum configuration of an 80286 with 640K of memory and supports most RLL, MFM, ESDI, and IDE hard drives (with SCSI support being added). The system comes on a mere four disks and is quick to install, by UNIX standards. Coherent is priced at $99.95 with the Device Drive Kit an additional $39.

Mark Williams Company
601 N. Skokie Highway
Lake Bluff, IL 60044
(800) 627-5967

MAGIC TSR TOOLKIT

Quantasm Corporation has released an assembly-language source code template and toolkit for creating TSR programs. Programs that are created with the template can swap to EMS or to hard disk, while keeping only 2 to 4K of the TSR in memory. The template prevents multiple installation and allows uninstallation, reading and writing from multiple files, enabling/disabling of individual hot-keys, redefining of hot-keys, de-activation, and full keyboard and mouse support.

Full assembly-language source code which is compatible with Microsoft's MASM 5.x, Borland's TASM, and SLR Systems' OPTASM is provided. The package includes a manual with a TSR tutorial and on-disk examples of each feature. The Magic TSR Toolkit is priced at $199.95.

Quantasm Corporation
19855 Stevens Creek Blvd.
Cupertino, CA 95014
(408) 244-6826

X VISION FROM GSS

Graphic Software Systems (GSS) announces their exclusive U.S. distributorship of XVision software, which is manufactured by VisionWare of Leeds, U.K. XVision allows PC users to display multiple X Window System applications alongside multiple Microsoft Windows applications.

XVision, a PC-based X server, is a Microsoft Windows application that displays multiple applications in X and MS-Windows without the need for a DOS hot-key mechanism. It also lets users cut and paste between the two environments. XVision can be networked to a variety of X hosts simultaneously (including UNIX and VMS) over a TCP/IP network. Users can select Windows or a host-based window manager, such as OSF/Motif or Open Look, to control X clients.

XVision is compatible with all versions of MS-Windows. Priced at $445, it runs on IBM-compatible PCs and supports all displays (EGA, 8514A, DGIS, and VGA) and input devices supported by MS-Windows.

Graphic Software Systems, Inc.
9590 SW Gemini Drive
Beaverton, OR 97005
(503) 641-2200

SOLID LINK SUPPORTS ZMODEM

Solid Software has released Solid Link, a telecommunications library for Modula-2 and C. The entire library is coded in assembly language for maximum efficiency. The library includes a complete implementation of the powerful ZMODEM protocol. The library features interrupt-driven buffered input and output, handles baud rates up to 112K using either hardware or software flow control, and supports eight COM ports with concurrent communication. A terminal module for ANSI and VT100 emulation and a scheduler module that supports preemptive multitasking are also included.

Solid Link incorporates most common file transfer protocols: XMODEM, XMODEM-1K, MODEM7/BATCH, YMODEM, YMODEM-G, ZMODEM, TeLink, and SEAlink (Kermit is being added). Handshaking supports XON/XOFF, DSR/DTR, CTS/RTS, and CTR-C/K handling. The library comes with a sample terminal program (in source form) that features terminal emulation, full support for all Solid Link protocols,

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

C CODE FOR THE PC

source code, of course

NEW!	Embedded File Manager (rommable code to control floppy and Winchester disks; MS-DOS file system; Posix-style program interface)	$1,500
	MS-DOS File Compatibility Package (create, read, & write MS-DOS file systems on non-MS-DOS computers)	$750
Windows!	C++/Views (C++ interface to MS Windows 3.0; over 60 classes; free broswer (no source); no royalties; Zortech C++ only)	$395
	GraphiC 5.0 (high-resolution, DISSPLA-style scientific plots in color & hardcopy)	$325
NEW!	Spontaneous Assembly (700 functions and macros, 750 page manual, 6 memory models, C bindings)	$300
NEW!	Victor Image Processing Library (EGA/VGA; TIFF/PCX/GIF/bin; multiple images; many filters; frame grabber & live video)	$250
	Time Series Management Tool (collection of routines for manipulating time series)	$250
	TurboTEX (Release 3.0; HP, PS, dot drivers; CM fonts; LaTEX; MetaFont)	$250
	db_File & db_Retrieve Bundle by Raima (B-tree and network database with SQL query and report writer; multi-user $475)	$245
	Install 3.05 (automatic installation program; user-selected partial installation; CRC checking)	$195
	Wendin-DOS Plus (self-bootable, multitasking, multiuser MS-DOS replacement; includes XTC editor)	$180
	QuickGeometry Library (large collection of mathematics, graphics, display & DXF subroutines for CAD/CAM/CAE/CNC)	$170
NEW!	Rogue Wave tools.h++ Class Library (collections, B-trees, file space management, time& date, persistent objects; complete user's guide)	$165
	Rogue Wave math.h++ Class Library (C++ overloadings for standard operators, matrix inversion & FFT; complete user's guide)	$165
	Turbo G Graphics Library (all popular adapters, hidden line removal)	$165
	Essential resident C (TSRify C programs, DOS shared libraries)	$165
	TurboGeometry (library of routines for computational geometry, Version 3.0)	$160
	WKS Library Version 2.01 (C program interface to Lotus 1-2-3, dBase, Supercalc 4, Quatro, & Clipper)	$155
	Hold Everything (spawn new programs; swap parent to EMS or disk; handles video, interrupts, & environment; returns error level)	$150
Windows!	RED (program development editor for Microsoft Windows; keystroke recording & playback; unlimited undo)	$150
	C Generator (generates C code to read & write file records defined with C structure syntax)	$150
	Cephes Mathematical Library (over 100 high-quality, double-precision scientific functions)	$150
	Vmem/C (virtual memory manager; least-recently used pager; dynamic expansion of swap file)	$140
Cheaper!	CBTree (B+tree ISAM driver, multiple variable-length keys)	$135
NEW!	C Communications Toolkit by Magna Carta (multi-port & co-processor board support, FAX, interrupt driven, emulations, xfer protocols)	$125
	C++ Object Library (virtual windows, I/O, lists, file free space management, keyed ISAM file I/O)	$125
	Envoy Communications Package (includes editor, script language, redefinable keys, help screens, menu-driven interface; Turbo C)	$120
	Kier CountryLib (date, time, measurement, numeric, and simple language conversion; helps build country-independent software)	$120
	SilverWare "C" EMM Library (60 interface functions to EMM Version 4.0)	$120
	TE Editor Developer's Kit (full screen editor, undo command, multiple windows)	$115
	PowerSTOR (upto a gigabyte of heap space on extended memory, expanded memory, and/or hard disk)	$110
	Wendin PCVMS and PCNX Shell Toolkits (operating system shells for VMS or Unix over DOS)	each $110
	B-Strings (dynamic string handling; cut, copy, paste, search, user input, etc.; non-fragmenting memory management)	$105
	PC/IP (CMU/MIT TCP/IP for PCs; Ethernet, Appletalk & NETBIOS drivers, RVD, gateways)	$100
Cheaper!	B-Tree Library & ISAM Driver (file system utilities by Softfocus)	$90
	Disassembler (8086/80286/80386/80C186 with 80287/80387/80C187 NPX)	$90
	Booter Toolkit (floppy disk bootstrap routines, DOS file system, light-weight multitasking, windows, fast memory management)	$85
	Otter 2.0 (beautiful theorem-prover by Bill McCune; includes manual & two books by Wos; complete starter kit)	$80
	MultiDOS Plus (DOS-based multitasking, intertask messaging, semaphores)	$80
NEW!	COMM-DRV (robust serial communications driver; COM1-COM25; INT 14H & EBIOS; FOSSIL & multi-tasker compatible; inc. file transfer)	$75
	eval() (C function to evaluate ASCII infix expression string; 17 built-in functions)	$75
	XT BIOS Kit (roll your own BIOS with this complete set of basic input/output functions for XTs)	$75
	Symtab/Ptree (general-purpose symbol table/parse tree construction and management package; specify Symtab or Ptree)	each $75
	XSPAWN (swap parent process to expanded memory or disk before running child)	$70
	Heap Expander (virtual memory manager using expanded memory, extended memory, and disk space)	$65
	Foundations-1 C++ Class Library (ASCII record I/O, bit arrays, exception handling, B-tree, persistent objects)	$60
	Coder's Prolog (Version 3.0; inference engine for use with C programs)	$60
	Async-Termio (Unix V compatible serial interface for MS-DOS; stty, ioctl, SIGINT, etc.)	$55
	Pascal P-Code Compiler & Interpreter or Pascal-to-C Translator (Level 0 ISO standard Pascal with some Level 1 features)	each $50
	Backup & Restore Utility by Blake McBride (multiple volumes, file compression & encryption)	$50
	Floppy TAR (TAR backup and restore on MS-DOS devices; direct access to non-standard devices)	$50
	SuperGrep (exceptionally fast, revolutionary text searching algorithm; also searches sub-directories)	$50
	OBJASM (convert .obj files to .asm files; output is MASM compatible)	$50
	CLIPS (rule-based expert system generator, Version 4.3; advanced manuals available at additional cost)	$50
	PCHRT (microsecond timers and delays, interrupt profiles, timer tick management)	$45
	Kier DateLib (all kinds of date manipulation; translation, validation, formatting, & arithmetic)	$45
	DES Encryption & Decryption (2500 bits/second on 4.77 MHz PC for on-the-fly encryption at 2400 baud)	$40
	FlexList (doubly-linked lists of arbitrary data with multiple access methods; specify C or C++)	each $40
NEW!	Window Boss & Data Clerk (windows, menus, data checking, on-line help, window drag, screen forms)	$40
	Virtual Memory Manager by Blake McBride (LRU pager, dynamic swap file, image save/restore)	$40
	Heap I/O (treat all or part of a disk file as heap storage)	$40
NEW!	Classy Forms (screen forms in both C and C++ versions)	$40
	RXC & EGREP (Regular Expression Compiler and Pattern Matching; RXC makes finite state machine from regular expression)	$35
Updated!	NIH Class Library (collection of basic C++ classes by Keith Gorlen; Version 3.0; uses C++ R2.0)	$35
	Bison & BYACC (YACC workalike parser generators; documentation; no restrictions on use of BYACC output)	$35
	PC-XINU (Comer's XINU operating system for PC)	$35
NEW!	PRINTF for ASM	$30
NEW!	Spell Pack (6 spelling programs & a hyphenator: Ispell, Microsp, Sp, Cspella, Spell, Hy, & the Dawg utilities)	$30
	REGX Plus (search and replace string manipulation routines based on regular expressions)	$30
	Translate Rules to C (code generator for rule-based systems)	$30
	GNU Awk & Diff for PC (both programs in one package)	$30
	6-Pack of Editors (baker's half-dozen public domain editors for use, study & hacking; includes microEmacs 3.10 & Stevie, a vi clone)	$30
Upgrade!	Crunch Pack (18 file compression & expansion programs)	$30
	CCALC (handy extended-precision calculator; real and complex models; many built-in functions)	$30
	PC-MAIL (UUCP mailer by Wietse Z. Venema; send, receive, and manage UUCP mail)	$25
NEW!	PERL for MS-DOS (C, sed, awk, and shell all rolled into one language; includes hardcopy docs)	$25
Updated!	FLEX (fast lexical analyzer generator; new, improved LEX; BSD Version 2.3 with docs)	$25
	Using C++ Library (the code from the book by Bruce Eckel and then some; Zortech 2.0 compatible)	$25
NEW!	VAL Linker by David Troendle (a complete linker; excellent for learning about linking)	$20
	Data	
	Moby Part-of-Speech (200,000 words and phrases described by prioritized part(s)-of-speech)	$120
	Moby Words (500,000 words & phrases, 9,000 stars, 15,000 names)	$80
	Smithsonian Astronomical Observatory Subset (right ascension, declination, & magnitude of 258,997 stars)	$60
	Dictionary Word List (234,932 words in alphabetical order)	$60
	U.S. Cities (names & longitude/latitude of 32,000 U.S. cities and 6,000 state boundary points)	$35
	The World Digitized (100,000 longitude/latitude of world country boundaries)	$30
	KST Fonts (13,200 characters in 139 mixed fonts: specify TEX or bitmap format)	each $30
	Interactive Computer Ephemeris (high-precision moon, sun, planet & star positions; USNO (no source) & Downey 4.15 (C source))	$30

The Austin Code Works
11100 Leafwood Lane
Austin, Texas 78750-3464 USA

Voice: (512) 258-0785
FAX: (512) 258-1342
E-mail: info@acw.com

Free surface shipping for cash in advance **For delivery in Texas add 7%** **MasterCard/VISA**

Circle 30 on Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

log file support, Procomm-compatible dialer files, EGA/VGA 43/50 line modes, and DOS shell support. The Modula-2 version supports compilers from Logitech, Stonybrook, and JPI. The C version of the library supports the large model of Microsoft C, Borland's Turbo C/C++, and Zortech C++.

Solid Software also offers Solid B+ Toolbox (a B+ tree database library) and Solid Windows for Modula-2. The B+ Toolbox supports dBASE III file access as well as variable length records. The B+ Toolbox is priced at $199 ($399 with source) and Solid Windows is $149. The Professional Windows version that includes a user interface library is $495. The Solid Link communications library is priced at $199 for C and Modula-2 ($497.50 with source code) and the C++ version is $248 ($546.50 with source code). All software comes with a 30-day money-back guarantee.

Solid Link
P.O. Box 36164
Cincinnati, OH 45236-6164
(513) 651-1133

CASE:PM VERSION 2.0

CASEWORKS is now shipping version 2.0 of CASE:PM, a case tool for developing OS/2 Presentation Manager applications. CASE:PM 2.0 incorporates several notable enhancements, such as extended code support for dialog controls such as push buttons and list boxes. Version 2.0 also supports OS/2's built-in help system and generates code for routines to execute in separate threads, allowing programmers to build applications that fully exploit OS/2's multitasking capabilities.

Unlike fourth-generation language development environments, CASE:PM generates code in standard programming languages (including C, C++, COBOL, and FORTRAN). As a result, CASE:PM does not require programmers to learn a proprietary language or special syntax. In addition, applications created with CASE:PM do not require special compilers or runtime systems in order to operate. As with all CASEWORKS products, the user owns the generated program codes, and there are no royalty or licensing fees.

While the developer uses CASE:PM to prototype an interface, CASE:PM monitors compliance with CUA (Common User Access) standards—a great help for programmers who are developing under IBM's Systems Application Architecture guidelines. Developers can selectively choose to override individual CUA violations as CASE:PM reports them, or disable the CUA monitor entirely.

Priced at $1995, CASE:PM operates under Presentation Manager. CASE:PM for C requires the Microsoft or IBM OS/2 Programmer's Tool Kit (version 1.2) and the corresponding C compiler, make utility, and linker.

CASEWORKS, Inc.
One Dunwoody Park, Suite 130
Atlanta, GA 30338
(800) 635-1577

INTEGRA SQL DATABASE

Coromandel Industries has released Integra SQL for Microsoft Windows, a powerful, compact SQL-based relational database product. Integra SQL allows developers to build large database applications for Windows, and is compatible with Microsoft Windows 2.1 and 3.0.

Integra SQL performs with ease as a well-behaved Windows application in the 640K environment. The database management system provides a choice of low- or high-level programming interfaces. The high-level interface is based on embedded SQL statements. The low-level interface allows the developer to manipulate tables directly and to perform record operations.

The package consists of a pre-compiler, client library, and local server, all running under Windows. It supports a language-based approach for building database applications, so developers can build applications by embedding SQL statements in a host language such as C. Integra SQL automatically handles complex data management, query optimization, and data integrity.

The database engine has flexibility because it is built on an Open Access Manager. Consultants, OEMs, and VARs can easily write gateways that allow Integra SQL to interface with other file managers and databases. Integra SQL is currently available under DOS, UNIX, SunOS, and Windows. Based on the ANSI SQL standard, Integra is also extensible and allows integration of user-defined functions into the engine. Coromandel offers consulting and database engine customization services to developers. Integra SQL for Windows is $695, with site licenses available.

Coromandel Industries, Inc.
108-27 64th Road
Forest Hills, NY 11375
(718) 997-0699

SOPHCO FORCE 2.1

Sophco has released the latest update of Force, a structured dBASE-language compiler. Force will produce native machine code from your source code, eliminating the need for a runtime interpreter. Force 2.1 allows programmers to create terminate-and-stay-resident (TSR) dBASE applications and includes comprehensive NetBIOS support. By compiling under Force 2.1, programmers can turn any dBASE application into a TSR. The NetBIOS support can be used to create a distributed processing environment in which portions of large tasks can be assigned to several network nodes, then collated at the file server.

The Force compiler validates and verifies all lines of source code at compile time, instead of at runtime. Force produces a single, executable file that can be distributed royalty-free. Sophco has reduced the price of its Force compiler from $695 to $199 until November 30, 1990. Force 2.1 comes with a 90-day unconditional money-back guarantee.

Sophco
P.O. Box 7430
Boulder, CO 80306
(303) 444-1542

CODE BASE VERSION 4.2

Sequiter Software has released Code Base Version 4.2, an updated version of their C library for database and screen management. Code Base gives C and C++ programmers dBASE or Clipper functionality and file compatibility. This C source code library provides complete support for both dBASE III and Clipper (both dBASE NDX and Clipper NTX index files).

Version 4.2 of Code Base will compile with C++ or ANSI C under DOS, OS/2, UNIX, or Microsoft Windows. It

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

also offers a customizable browse/edit facility. Though Code Base may be used in a single-user environment, the library has full support for multi-user applications under DOS networks, OS/2, and UNIX. The network support is also compatible with Clipper locking and unlocking protocol, allowing Code Base and Clipper applications to execute at the same time on a network.

Using the Code Base 4.2 library, application programs can be written that access data faster than dBASE IV or FoxPro. Once the data has been read, the full expressive power of the C language is available for processing. The source code is compatible with ANSI C. DOS C compilers directly supported include Microsoft's C (6.0 and older versions) and Quick C, Borland's Turbo C/C++, Zortech C++, and Watcom C-386. The portable version for UNIX adds source-code support for non-ANSI C compilers and uses Curses for screen management.

Both C source code and compiled object-module libraries are provided. No royalties or licenses are necessary for distributing applications built using these libraries. The DOS and OS/2 version is priced at $295, and the portable version is $495.

Sequiter Software Inc.
#209, 9644 - 54 Avenue
Edmonton, Alberta
Canada T6E 5V1
(403) 448-0313

TOPAZ 3.0 FOR PASCAL

Software Science has released version 3.0 of its popular database management software, TOPAZ, which allows programmers to write applications in Pascal using standard dBASE files and syntax. TOPAZ 3.0 includes memo field support, a programmable text editor library, multi-user support for all popular LANs, tag lists and additional searching tools, and the ability to create and maintain "virtual" database files. In addition, almost 100 new commands have been added, such as **peek**, **poke**, **search**, and **list**.

Multi-user support in TOPAZ 3.0 is unusual in that locking records does not prevent other users from reading or reporting on those records. Locking methods in TOPAZ are especially robust: the TOPAZ library is designed to survive node and server power interruptions with all data intact.

Data can now be managed in TOPAZ via a "virtual" database model. Essentially a sophisticated linked-list manager, TOPAZ 3.0 allows programmers to handle a variety of both fixed and variable-length records with exactly the same syntax as that available in dBASE dialects. Data such as text, graphics, laboratory data, and audio sound bites can be stored and handled as easily as any conventional dBASE file.

TOPAZ requires the Borland Turbo Pascal compiler (4.0 or later) or Microsoft's QuickPascal compiler (1.0 or later). TOPAZ comes with CREATE and BROWSE utilities, a code generator that automatically makes complete database management programs for both single-user and multi-user applications, and a dBASE-like report generator. TOPAZ does *not* require dBASE. Source code to the TOPAZ library is available to registered users at no cost. Single-user TOPAZ is priced at $99 and multi-user is $149. Upgrades from earlier versions are also available.

Software Science, Inc.
100 Valley Drive
Brisbane, CA 94005
(415) 467-6840

MULTIBASE DATABASE SERVER

MultiBase II, a multi-user, multitasking dBASE-compatible database engine, is available from Cykic Software. MultiBase II creates an operating environment that transforms single-user applications into true multi-user applications without rewriting code. Users on all workstations in the MultiBase environment are able to access all data and peripherals.

The MultiBase II environment is created by loading MultiBase directly from DOS to any 286, 386, or 486-based PC. The MultiBase engine then acts as a file server to support up to 30 client workstations attached to that PC via serial ports or ARCnet adapters and cables. No additional hardware or software is required.

MultiBase client workstations can also be set up at remote sites via modems. Developers can use MultiBase to connect their PC to a user's PC in order to update and maintain applications. MultiBase also allows each user to run multiple jobs at the same time, including background printing, access to multiple modules, and more. Developers can set up several jobs to run simultaneously within one application.

MultiBase II can edit, compile, and debug dBASE programs. In fact, commands can be mixed and matched from dBASE, Clipper, or FoxBASE+ dialects. Multibase II is priced at $649.

Cykic Software
3944 Third Avenue
San Diego, CA 92103
(800) 544-4620

PARAGEN C CODE GENERATOR

Concept Dynamics has now released PARAGen, a C code generator for the Paradox Engine. PARAGen produces C source code specific to a user's application which contains calls to the Borland Paradox Engine. Using Paradox tables previously defined by the user, PARAGen examines the data structure of the tables and produces code to perform a variety of operations on those tables.

Using the Paradox Engine, functions that are generated by PARAGen will open, close, create, and empty Paradox tables. Record-level operations include full and partial searches, first, last, next, and previous record fetches, and field searches. Append, insert, and delete record operations are also provided.

Because it automates database access code generation, PARAGen allows developers to spend more time writing application code. Though the Paradox Engine provides the routines necessary to perform table operations, these are generic functions that have no knowledge of the tables defined by the application developer. PARAGen provides the missing link that ties an application to the Paradox Engine. PARAGen sells for $99.

Concept Dynamics, Ltd.
1147 South Euclid
Oak Park, IL 60304
(708) 524-2814

SUPERFUNCTIONS UPGRADE

Greenleaf Software has released an upgrade to its SuperFunctions C library. This library furnishes C programmers

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

with a variety of complex services for DOS platforms, including functions for expanded memory interface, critical error handlers, Microsoft mouse interface, and dozens of functions to create and manage text windows. SuperFunctions gives programmers a number of functions for time and date manipulation, along with timers and alarms.

New with this release are additional functions to manage windows, such as the capacity to paint windows with shadows, scroll text within the windows, and write text with a wide array of different color and intensity attributes.

SuperFunctions now supports additional C compilers: Watcom C, Zortech C++, and TopSpeed C. It continues to support Turbo C 2.0, Turbo C++, Lattice C, Quick C, and Microsoft C 6.0. SuperFunctions can also take advantage of the increased speed of the Microsoft C fast call convention. This strategy allows some parameters to be passed to the routine by way of the CPU registers.

SuperFunctions offers a complete set of routines designed to interface applications with Microsoft-compatible mouse drivers. This includes detection of the mouse, mouse cursor setup and maintenance, status, position, and click detection. The mouse interface also has both a graphic screen interface and text modules using either software or hardware cursors.

SuperFunctions costs $299, and the upgrade price is $45. A free demo disk is also available.

Greenleaf Software, Inc.
16479 Dallas Parkway, Suite 570
Dallas, TX 75248
(800) 523-9830

UPGRADE FOR PCX TOOLKIT

Genus Microprogramming has now released version 4.0 of their PCX Programmer's Toolkit. The Toolkit allows IBM PC software developers to easily create applications with the ability to manipulate PCX-format graphic images.

Version 4.0 also marks the introduction of Genus's new GX Development Series. Unlike other graphics libraries that use memory-resident kernels, all of the products within this series share a centerpiece of common functions that are linked directly into the developer's application.

Programs using libraries that are members of the Genus GX Development Series can save on code overhead by sharing kernel functions between libraries. Only the required kernel functions will be linked into the programs. The kernel is responsible for all display adapter interfacing, memory allocation (whether it is conventional, expanded, or disk-based), and virtual bitmap support, all at assembly language speed.

In addition to the new kernel system, Genus has added several new functions to the PCX Toolkit. Sizing and scaling of images and enhanced virtual (large, off-screen) buffer functions have been incorporated. Virtual buffers can be created in conventional or expanded memory, or on disk. Support for more Super VGA chipsets has been added, including support for up to 800x600 resolution in 256 colors. Genus has also developed a Super VGA detection routine to help standardize the diverse Super VGA modes.

The Toolkit supports all display modes of Hercules, CGA, EGA, VGA, and Super VGA display adapters, and contains a comprehensive set of compiler interfaces. These interfaces include Microsoft C, Quick C, Turbo C/C++, Lattice C, Microsoft BASIC, Microsoft Pascal, Turbo Pascal, Microsoft FORTRAN, MASM, TASM, and Clipper. The Toolkit includes over 75 routines, written entirely in assembly language for maximum speed. Libraries that are directly linkable to the user's program are provided, in addition to Quick Libraries for Microsoft-integrated compilers.

The price of the PCX Toolkit is $249. The source code is available for an additional $350 when purchasing the package. In addition, upgrades from earlier versions are available.

Genus Microprogramming, Inc.
11315 Meadow Lake
Houston, TX 77077
(800) 227-0918

SUBSCRIBE NOW

U.S. Postal Service
STATEMENT OF OWNERSHIP, MANAGEMENT AND CIRCULATION
(Required by 39 U.S.C. 3685)

1A. Title of Publication	1B. PUBLICATION NO.	2. Date of Filing
Programmer's Journal	0 7 4 7 5 8 6 1	9-30-90
3. Frequency of Issue	3A. No. of Issues Published Annually	3B. Annual Subscription Price
Bi-Monthly	6	$24.00

4. Complete Mailing Address of Known Office of Publication (Street, City, County, State and ZIP+4 Code) (Not printers)
Po Box 70167 Eugene Lane OR 97401-0121

5. Complete Mailing Address of the Headquarters of General Business Offices of the Publisher (Not printer)
Po Box 70167 Eugene, OR 97401

6. Full Names and Complete Mailing Address of Publisher, Editor, and Managing Editor (This item MUST NOT be blank)
Publisher (Name and Complete Mailing Address)
Elizabeth Oakley Po Box 70167 Eugene, OR 97401
Editor (Name and Complete Mailing Address)
M Steven Baker Po Box 70167 Eugene, OR 97401
Managing Editor (Name and Complete Mailing Address)

7. Owner (If owned by a corporation, its name and address must be stated and also immediately thereunder the names and addresses of stockholders owning or holding 1 percent or more of total amount of stock. If not owned by a corporation, the names and addresses of the individual owners must be given. If owned by a partnership or other unincorporated firm, its name and address, as well as that of each individual must be given. If the publication is published by a nonprofit organization, its name and address must be stated.) (Item must be completed.)

Full Name	Complete Mailing Address
Oakley Publishing Company	Po Box 70167 Eugene, OR 97401
Elizabeth Oakley	Po Box 70167 Eugene, OR 97401
James Oakley	Po Box 70167 Eugene, OR 97401

8. Known Bondholders, Mortgagees, and Other Security Holders Owning or Holding 1 Percent or More of Total Amount of Bonds, Mortgages or Other Securities (If there are none, so state)

Full Name	Complete Mailing Address
none	

9. For Completion by Nonprofit Organizations Authorized To Mail at Special Rates (DMM Section 423.12 only)
The purpose, function, and nonprofit status of this organization and the exempt status for Federal income tax purposes (Check one)
(1) ☐ Has Not Changed During Preceding 12 Months (2) ☐ Has Changed During Preceding 12 Months (If changed, publisher must submit explanation of change with this statement.)

10. Extent and Nature of Circulation (See instructions on reverse side)	Average No. Copies Each Issue During Preceding 12 Months	Actual No. Copies of Single Issue Published Nearest to Filing Date
A. Total No. Copies (Net Press Run)	30,080	29,820
B. Paid and/or Requested Circulation 1. Sales through dealers and carriers, street vendors and counter sales	9,036	12,823
2. Mail Subscription (Paid and/or requested)	16,856	15,814
C. Total Paid and/or Requested Circulation (Sum of 10B1 and 10B2)	25,892	28,637
D. Free Distribution by Mail, Carrier or Other Means Samples, Complimentary, and Other Free Copies	1,061	734
E. Total Distribution (Sum of C and D)	26,953	29,371
F. Copies Not Distributed 1. Office use, left over, unaccounted, spoiled after printing	864	449
2. Return from News Agents	2,263	0
G. TOTAL (Sum of E, F1 and 2—should equal net press run shown in A)	30,080	29,820

11. I certify that the statements made by me above are correct and complete
Signature and Title of Editor, Publisher, Business Manager, or Owner
Elizabeth D. Oakley
(See instructions on reverse)

PS Form 3526, Feb. 1989

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

PJ BILLBOARD

FIGHT PIRACY & PROTECT YOUR PROGRAM $$$'s!

Since 1986, companies worldwide have been choosing Az-Tech security products. If you demand the strongest protection available, why not choose one of these "proven leaders":

- **EVERLOCK** Copy Protection
- **EVERTRAK** Software Security
- **EVERKEY** Hardware "Key" Software Security

For IBM and Compatibles.
30 day money back guarantee.
Free info and demo disk available.

Az-Tech Software, Inc.
305 East Franklin
Richmond, MO 64085
(800) 227-0644 (816) 776-2700
FAX (816) 776-8398

Circle 51 on Reader Service Card

MORE MEMORY FOR C PROGRAMMERS

PowerSTOR makes system memory resources (LIM 3.2/4.0 expanded memory, AT style extended memory, hard disk or any combination) available to your software as STORheaps which can be allocated and de-allocated dynamically. Up to a gigabyte of PowerSTOR memory can be used as one huge STORheap or many smaller STORheaps. Uses only 15K.

CALL FOR A FREE DEMO DISK
1-800-621-5663
Acme Software
503-235-2816
P.O. Box 40734, Portland, OR 97240

Contact this advertiser directly

EZ-COPY PLUS

Duplicates Disks Fast!

•2X+ faster than DOS •Read diskette once, then...•Quickly, accurately mass duplicate 5.25" & 3.5" disks (360K, 720K, 1.2M, 1.44M) on your own PC/XT/AT/etc. © •Supports up to 4 drives with automatic start and switching. •Formats, copies, verifies, etc. • Fast partial copies too. •Even faster with pre-formatted disks. •Serializes disks and/or labels, in 1 smooth operation. •Save disk images to HD, more... •Replaces hardware worth $1000s.

Only $139 +S&I CP, for 1 machine.
Only $495 +S&I NCP, for upto 10 mach.

S&I: $6 USA48; $10 N.Amer; $22 Foreign

EZX, 917 Oakgrove, #101, Houston,TX 77058
Info:713.280.9900 •BBS:8180 •FAX:0525
Order (D,V,M,AX), Catalog Toll Free
1☎800•359•953•9 xPJ1190

Circle 352 on Reader Service Card

FREE CATALOG!

LATEST AWARD BIOS
PC/XT, 286, 386

COMPILED BASIC TOOLS
Libraries, Utilities
for QuickBASIC 4.5

HARDWARE UPGRADES
for PCs & PC/XTs

(800) 423-3400 - (412) 782-0384
KOMPUTERWERK INC.
851 Parkview Blvd.
Pittsburgh, PA 15215

Circle 155 on Reader Service Card

"QUICK & CLEAN" COBOL

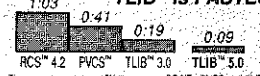
The COBOL Report Activator (COBRA):

- Generates efficient, structured COBOL programs
- Is easy to learn and use
- Accepts your COBOL code and COPY members
- Produces audit trail reports
- Up to 5 output files per program
- Provides report options & site defined defaults
- Is available for Mainframe & PC

COBRA
Systems Software
David Olivadoti, P.O. Box 750817
Petaluma, CA 94975
(707) 763-5123

Circle 281 on Reader Service Card

TLIB™ is FASTEST!



Times are to update a 45K library on a PC/XT. PVCS and TLIB 3.0 are from Sept 87 *PC Tech Journal*. MKS RCS 4.2 and TLIB 5.0 are newer.

TLIB™ is *BEST!*

"Do not be fooled by the fact that this is the least expensive of the five packages reviewed here - TLIB has features and power to spare" John Rex, **Computer Language**

"TLIB is a great system" J. Vallino, **PC Tech J**

• **Full-Featured Version Control for Software Professionals.** Check-in/out locking. Branching. Keywords. Wildcard and list-of-file support. Can merge parallel changes and undo intermediate revisions. Network and WORM support. **Mainframe compatible** deltas for Pansophic, ADR, IBM, etc.. Integrates with **Opus™ MAKE** & **Slick™ MAKE**.

MS-DOS $139, OS/2 $195 + shipping Visa/MC 5 station LAN license $419 (OS/2 $695), call for other sizes

BURTON SYSTEMS SOFTWARE
PO Box 4156, Cary, NC 27519 (919) 233-8128

Circle 288 on Reader Service Card

Why you want BATCOM!

BATCOM is a batch file compiler that compiles your ".bat" files to ".exe" files to make them faster, more professional, and more capable. **BATCOM** extends DOS with new commands so you can read keyboard input, perform arithmetic, use subroutines, and much more. In addition, **BATCOM** protects your source code, and you can distribute your compiled programs without royalties. Only **$59.95**

ORDER TODAY!

Wenham Software Co.
5 Burley St.
Wenham, Ma. 01984
(508) 774-7036

Circle 320 on Reader Service Card

FREE TRIAL

Pinnacle File Manager

We guarantee immediate productivity gains from using our C file manager library. Simple table-and-column view of data with advanced B-Tree indexing and hi-speed cache frees your brain to solve application-specific problems.

Vermont Database Corp.
RR3 Box 6516
Stowe, VT 05672
800-822-4437

Contact this advertiser directly

***PJ* CODE IS AVAILABLE!**

All the source code from the articles in this issue is available on disk from

Programmer's Journal
P.O. Box 70167
Eugene, OR 97401-0110
800-234-0386
503-747-0800

Code disks are $10.00 each. Please see page 80 for ordering information.

PJ Code DiskScriptions are available for $39.95. See page 16 to order.

The code listings are also available for downloading from several bulletin board systems around the country.

Contact this advertiser directly

CodePrint for C
SOURCE CODE REFORMATTER and PRETTY PRINTER

CodePrint is a full-featured menu-driven source code reformatter and pretty printer with more than twenty customization features.

- syntax-driven auto-indent
- comment alignment
- adjustable tab spacing
- program flow lines
- line edit for white space
- on-line help
- batch processing
- print spooling

For IBM PC/AT/XT/PS2s and compatible computers and IBM or EPSON configurable printers. 30-day money-back guarantee. **Price $49.**

North River Software
PO Box 36335, Indianapolis, IN 46236
317-240-8445

Circle 71 on Reader Service Card

The $25 Network

Try the 1st truly low cost LAN

- Connect 2 or 3 PCs, XTs, ATs, PS/2s
- Uses serial ports and 5 wire cable
- Runs at 115K baud, up to 90 feet
- Transfer 8500 bytes per second (ATs)
- Runs in background, totally transparent
- Share any device, any file, any time
- Needs only 14K of ram
- Replace all file transfer software
- $25 per network, NOT PER NODE!
- OVER 15,000 SOLD WORLD WIDE

Skeptical? We make believers!

iMODES Information Modes
Drawer F, Denton, TX 76202
817-387-3339 Technical
1-800-628-7992 Orders

Circle 348 on Reader Service Card

OPT-TECH SORT/MERGE

Extremely fast Sort / Merge / Select utility. Run as an MS-DOS command or CALL as a subroutine.
Supports most languages and filetypes including Btrieve and dBase. Unlimited filesizes, multiple keys and much more!

MS-DOS **$149.**
OS/2, UNIX **$249.**

Opt-Tech Data Processing
P. O. Box 678
Zephyr Cove, NV 89448
(702) 588-3737

Circle 16 on Reader Service Card

Offer your product directly to Professional Programmers:

Placing an ad in **PJ BILLBOARD** is an easy, low-cost way to reach potential buyers. Regular listings include a headline, company information, and approximately 400 characters for your message. Display ads are 1 5/8" wide by 2 3/8" high. For more information or to reserve a space, call our ad department at **503-747-0800.**

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

DO IT YOURSELF!

XenoCopy-PC

PC-DOS program lets your PC
Read/Write/Format
over 350 formats **$79.95**

XenoFont

high quality text screen printouts
$49.95

+$5.00 S/H + Tax if CA.

XenoSoft • (415) 644-9366
2210 Sixth Street,
Berkeley, CA 94710

Circle 123 on Reader Service Card

sPORTt
serial PORT tester

sPORTt is a serial port tester for people who don't want to be data communications experts, but use or write software for serial devices, such as plotters, mice, digitizers, modems, serial printers, or computer to computer links.

- displays all data going in and out in ASCII, decimal, hex, binary
- lets you choose handshaking lines
- sends/receives disk files
- automatically runs port diagnostics

Order today only $89.95!

WHS
3037 Grass Valley Hwy #8201
Auburn, CA 95603
916-885-2480

Circle 329 on Reader Service Card

PC TIMER TOOLS
New Version!

PCHRT is the definitive answer to execution profiling, precision timing and timer tick interrupt management in a PC/MSDOS environment. Over 50 functions manage multiple microsecond timers, generate precision delays, insert timers in most any interrupt, and run the timer tick interrupt at a user specified rate. Utilities to time execution and profile interrupts of any .EXE. Supports TC, TP, and MSC. Full library source included. $49.95 ppd USA, elsewhere add $4.00. VISA/MC.

RYLE DESIGN
PO Box 22, Mt. Pleasant MI 48804
517-773-0587

Circle 210 on Reader Service Card

IS 640K JUST NOT ENOUGH?

The **ATLAS MEMORY BOARD** adds 64K of DOS USABLE memory to the base 640K, making the total CONVENTIONAL memory 720,896 bytes. It does not use any memory resident drivers, and is compatible with all known applications. The same board works in any PC, XT, AT or 386 compatible, with monochrome graphics or CGA video. Memory added is NOT extended or expanded memory. The Atlas memory board has been certified by both NOVELL and 10 Net. Most orders shipped same day.

ATLAS MEMORY BOARD
15 Day Free Evaluation **$149.00** *Qty 1 Quantity Discounts Available*

M/C 1-800-535-6712 VISA
9AM - 9PM MON-SAT
NATIONAL DISCOUNT COMPUTERS
702-459-0049 NV Residents & Tech Support

Circle 107 on Reader Service Card

QUICKBASIC PDS 7/NETWORK
B+ TREE FILE INDEXING

Maintain B+ Tree file indexes in your QuickBASIC/PDS 7/NETWORK programs with INDEX MANAGER. Access files randomly by full or partial key, or sorted forward or backward. One module performs all functions, making IM very easy to use. Only indexes managed! You retain full control over your data. Assembler code and cache buffers make IM very fast. Includes sample programs and User's Guide. Only $59 (+tax in CA). PDS 7 version $99. Network version $250.

CDP Consultants
1700 Circo Del Cielo Drive
El Cajon, CA 92020
619-440-6482

Circle 335 on Reader Service Card

Software Marketing

The *Software Success Reference Book (1987-1988)* is a **MUST READ** if you want to market your software products successfully. Written by David H. Bowen, publisher of *Software Success,*™ the monthly newsletter on successfully running a software business. The *Reference Book* is a 268-page guide organized by topic. Covers Lead Generation, Promotion, Pricing, Distribution, Support, etc. **Only $25.** Check or Credit Card (VISA/MC/AEX).

100% Money Back Guarantee

Software Success
PO Box 9006, San Jose, CA 95157
(408) 446-2504
FAX (408) 255-1098

Circle 67 on Reader Service Card

Developer's Office Management System

Easy to use computerized system to organize, operate and control your software business.

Major features include:
- User and Dealer indexed database
- "Bug" look-up DB for software support
- Invoicing capabilities
- Utilities i.e., serial # embedding program
- Directory of packagers, suppliers, etc.
- Directory for product releases
- Lots of reports and other features
- Free 900 number to support your software

$69.00 (+ tax in Texas)
(VISA/MasterCard)
(512) 441-2615 (information)
(800) 992-9917 (sales only)

S.L.A.M. Software
3202 W. Anderson Ln. Ste. 208-176
Austin, TX 78757-1022

Circle 12 on Reader Service Card

UPPER DECK FORTH
$49

Based on the Forth-83 Standard. Fully segmented architecture. Fast compile, no link required. Uses normal DOS files. Includes editor, assembler, decompiler and debugger. Turnkey application support. For PC's and compatibles, 256K, DOS 2.0 or later. Add $3 for shipping and handling (outside USA $15). CA residents add sales tax.

UPPER DECK SYSTEMS
P.O. Box 263342
Escondido, CA 92026
(619) 741-1075

Circle 24 on Reader Service Card

The EDitor

Powerful **UNIX**-style programmer's editor. For use on IBM or compatible. Comprehensive manual included. A solid value at only $74.95 (US).
Available from:

Blue Jay Systems
PO Box 608
Mount Vision, NY 13810
607 293-8007

Circle 11 on Reader Service Card

PARADOX POPUP MENUS

Give your Paradox application an interface similar to PC Tools. Includes popup menus, dialog boxes, input/output boxes. Fast, tight, EASY. Includes PAL source code. Runtime and network ready. Only $45 plus $5 S/H.

PARADOX FIELD WRAPPING

Wrap text automatically down a Paradox field column. No more 256 character limit! This is a quick PAL procedure, and does NOT shell to DOS like competing products that require RUN BIG. Only $45 plus $5 S/H.

Hardware: 10 Mhz 8088 XT turbo/640k or better recommended.

Robin Rowe Software
1260 N. Dearborn, #310
Chicago, IL 60610-2240
312-280-7693

Paradox

Circle 74 on Reader Service Card

COMPUTER BOOKS

at the *lowest prices* of the year. Biggest savings on hundreds of closeouts for programmers and application users.

COMPUTER BOOKS
P.O. Box 70746
Sunnyvale, CA 94086

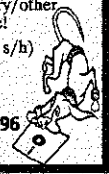
Circle 340 on Reader Service Card

INTEL® 80386 MACRO DISASSEMBLER

MD86 (Masterful Disassembler) is the most comprehensive macro disassembler on the market. Interactive by design not after-thought. Features include:

- ▸ User defined macros with arguments
- ▸ Auto code/data separation
- ▸ Auto commenting and label naming
- ▸ Command menus and help screens
- ▸ Customizable disassembly options
- ▸ Comprehensive, indexed, manuals
- ▸ Fully supports 8086/87 thru 80386/87
- ▸ COM/EXE/SYS/memory/other
- ▸ Cross reference and more!

Still only $67.50 + tax ($1.50 s/h)



C.C. SOFTWARE
1907 ALVARADO AVE.
WALNUT CREEK, CA 94596
(415)939-8153

Circle 132 on Reader Service Card

Coming Next Issue:

Graphics & Windowing

January/February 1991 Vol. 9.1
Advertising space reservations are due:
November 2, 1990
Materials are due:
November 6, 1990

Please call 800-234-0386
Ask for Jay or Robin in the advertising department to reserve your space.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

ADVERTISER INDEX

Reader Service	Advertiser Name	Page Number
*	Acme Software	93
345	Alphax Systems, Inc.	87
118	Artisoft Inc.	9
30	Austin Code Works, The	89
51	Az-Tech Software, Inc.	93
14	Blaise Computing, Inc.	51
2	Blaise Computing, Inc.	53
15	Blaise Computing, Inc.	55
11	Blue Jay Systems	94
3	Borland International	C4
288	Burton Systems Software	93
132	C.C. Software	94
321	Catenary Systems	28
335	CDP Consultants	94
53	Cnapse	95
281	Cobra Systems Software	93
39	Coleman Softlabs	5
340	Computer Books	94
229	Disk Software, Inc.	27
31	Double R Software	52
69	Evergreen Case Tools	34
352	EZX	93
261	Genus Microprogramming, Inc.	8
*	Gimpel Software	60
18	Goin Places Software	52
56	Greenview	35
334	Hamilton Laboratories	52
42	Harvard Softworks	18
348	Information Modes	93
46	InSync Software	95
50	INTEL	77
211	Kadak Products, Ltd.	66
295	Knowledge Dynamics Corp.	24
155	Komputerwerk	93
63	Laboratory Microsystems, Inc.	87
44	Laney Computer Systems, Inc.	69
246	Magee Enterprises, Inc.	68
40	Microsystems Software	29
272	Midnight Engineering	69
121	MISOSYS, Inc.	12
265	Mix Software	51
94	MMC AD Systems	49
109	Nanosoft	73
107	National Discount Computers	94
71	North River Software	93
160	Nu-Mega Technologies	3
16	Opt-Tech Data Processing	93
254	Opus Software	17
208	PC Tech, Inc.	28
36	PC Tech, Inc.	73
6	Periscope Company, Inc., The	87
*	***PJ* Bulletin Board**	41
*	***PJ* Mail Order Store**	80
279	Pocket Soft, Inc.	15
54	Programmer's Corner, The	17
48	Programmer's Warehouse	11
60	QNE International	54
*	Quaid Software Limited	62
252	Quantasm Corporation	85
147	Quarterdeck Office Systems	C2
147	Quarterdeck Office Systems	1
296	Que Corporation	81
20	Rational Systems, Inc.	63
59	RJSwantek & Associates	7
74	Robin Rowe Software	94
210	Ryle Design	94
12	S.L.A.M. Software	94
26	Sage Software, Inc.	59
32	Sandpaper Software	25
65	Schroff Development	22
269	Seaurer Software Inc.	33
165	SLR Systems	19
167	Software Development '91	13
67	Software Success	94
110	Softway, Inc.	23
47	Solution Systems	C3
330	Stanwood Associates, Inc.	26
68	Sub Systems	85
289	TechnoJock Software, Inc.	95
166	Tool Makers, The	41
327	Tools & Techniques, Inc.	24
276	Tri-Technology Systems, Inc.	56
72	TSR Systems	39
25	TurboPower Software	78
24	Upper Deck Systems	94
125	V Communications, Inc.	57
*	Vermont Database Corp.	93
304	Visitech Software	27
320	Wenham Software Co.	93
100	Western Wares	56
329	WHS	94
123	XenoSoft	94

When contacting any of the above advertisers, please tell them you saw their ad in ***Programmer's Journal*** and that you appreciate their continuing support of this magazine. The advertisers marked with an asterisk (*) prefer to be contacted directly. *PJ*'s Advertiser Index is provided as a free service, and ***Programmer's Journal*** cannot be held responsible for omissions or errors. ***Programmer's Journal*** is not responsible for errors in advertisements.

Be in (real-) time

Don't reinvent the wheel! Get the Synapse Real-Time eXecutive (SynRTX) source code library to write OBJECT-BASED MULTITASKING applications in C or C++.

- Four distinct classes of functionality: Task, Device, Cpu, and Timeout
- Complete scheduling control
- Serious real-time (no use of MS-DOS)
- Sample of real-time applications to get started (included in demo disk)
- Comprehensive user's manual
- Full library source code included
- No royalties

Demo disk & user's manual $75
SynRTX library source code $2475
Demo disk and user's manual are deducted if library source code purchased. Add $5 for US/Canada and $15 for international shipping and handling. All funds must be U.S. funds.

Cnapse Tel: (514)447-7221 Fax: (514)447-7297
11 Laurier, Suite 103, Chambly, Qc, Canada J3L 9Z7

Circle 53 on Reader Service Card

TechnoJock SOFTWARE

If you program in Turbo Pascal, use TechnoJock's Turbo Toolkit for menus, windows, input screens, and much more. $49.95 plus shipping & handling.

"With this Toolkit everyone becomes a TechnoJock."
Turbo User Group

TechnoJock Software, Inc.
PO Box 820927
Houston TX 77282
(713) 493-6354

Circle 289 on Reader Service Card

ATTENTION Software Writers!

Is your software innovative, usable and needed? Are your applications practical, thorough and well tested? Is your documentation well organized and understandable? InSync, a division of an established international software distributor, is evaluating new and innovative software to publish and market domestically and internationally. We are seeking new off-the-shelf MS/PC-DOS software applications (no accounting or inventory, please). ORIGINALITY IS A MUST for all submissions.

Please call our Evaluations Department to discuss submission procedures.

InSync software

1-800-783-7053

Circle 46 on Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

OFFRAMP

Oh say can you C?

By Hal. W. Hardenbergh

Computerdom gets a new fashionable language every decade. In the beginning was machine code. Then came assembly language, FORTRAN, Pascal, and now C. I don't know what language will be fashionable ten years from now, but I do know it won't be C.

Why? Because C is spectacularly ill-suited for the latest, fastest microprocessors. I'm not referring to the 80486 or even the 68040, but to the i860 and its cousins the 320C30, DSP32C, and 96002. All of these are micros with incredibly high floating point performance. Using single-precision FP, the peak performance of these machines is 80, 33, 25, and 40 MFLOPS, respectively. The i860 is a more general-purpose machine which can also execute 60 MFLOPS double precision; the other three are strictly single-precision micros that are primarily marketed as digital signal processing (DSP) chips.

These four chips get their high performance via an advanced form of pipelining that is fundamentally incompatible with current compiler technology, to which the C language is pretty much tied. (I wrote an incrementally compiled BASIC for the Atari ST, so I know a little bit about this.) Worse, the C language is hostile to floating point. You don't have to take my word for it: check pages 14 and 21 of *Numerical Recipes in C*. I haven't learned much about C because most of my programming involves some sort of mathematical modeling, which means floating point.

Any affordable technology which offers a significant performance advantage *will* be used. Some folks have a hard time believing that. Remember when no single operator would ever be able to use the incredible power of a 6-MHz 286, so the original PC/AT came with built-in capability to support several users? Then remember when the 386 came out and folks said nobody would ever need that much horsepower? There will be about nine million 386-style "nobodies" this year!

Nobody needs double-digit (that is, 25 to 80) MFLOPS, right? Personal MFLOPS? I can't speak for "nobody,"

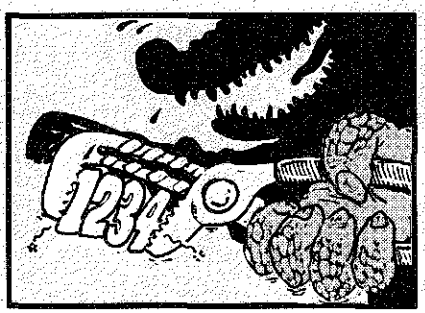


ILLUSTRATION BY JIM CARPENTER

but *I* need that level of performance. Sitting in my living room as I type this is a yet-unopened box which contains an AT-compatible board with a 25-MFLOP DSP32C chip and 640K of fast static RAM. It comes with an assembler and with software support from DOS via Quick C, so I'll have to learn more about C. I bought Microsoft's new Quick C with Quick Assembler so now I have two assemblers, one for the x86 and one for the DSP32C. (The board also comes with a math library in source form to help the user to get started with DSP32C assembly language.)

Of those four double-digit MFLOP chips, the DSP32C is the only one specifically designed to have a low price, which is also why it has the lowest performance. I can write software in assembly, and I did write a check for $1,222 to Symmetric Research in Kirkland, WA. Any affordable technology which offers a significant performance advantage *will* be used.

Last year I co-authored an article on artificial neural nets (ANNs) which *PJ* published. That article concluded by asserting that ANNs were an essentially infinite 'sink' for FP performance, and the assertion has proven regrettably true. I've been trying to get around the problem by keeping my three Atari STs going 24 hours a day much of the time. Each ST runs (with an assembly code boost) at the same speed as my 386/387 machine using Quick BASIC. I've been doing this for about 18 months.

I recently stumbled onto a method of considerably improving the accuracy with which an ANN can be made to do mapping. (Pattern recognition and mapping are the two things a multi-layer perceptron is good at.) At the time, I was trying to find a way to speed up the training; the considerable improvement in accuracy came as a complete surprise. The thing is, the method doesn't always work. When it works—which is 10% to 30% of the time—it gives *substantially* improved results.

I need to run a lot of trials to gather statistical data on this new method. Each trial requires 20 minutes of computer time using an ST or 386/387. One computer provides 72 results per 24 hours. I now have the results of about 2,500 trials. What I need to do is get 10,000 or so results and then modify the algorithm and get 10,000 more results and then do a statistical comparison. I'll need to repeat this sequence dozens of times. See the problem?

My guess is that I'll get a 100- to 300-times speedup using the new accelerator board. So I'll be able to do in a day what would take my three STs a month or more, and in one month what would take the STs several years. I *need* double-digit FP performance to enable me to investigate this method of training ANNs. (Although the method is not secret, space here is limited.)

Compilers have to use these new chips in "scalar mode," which drops the performance to single-digit MFLOPS. Most folks are *not* going to write code in assembly. We're going to need a new compiler technology to take full advantage of these chips, and that means a new language. What language? I dunno. Almost certainly not Occam.

Vectorizing compilers for these chips have been announced. They will, of course, be shipped Real Soon Now. Current work seems to be focussed on extensions to FORTRAN.

Before any of you send me letters angrily asserting that C will last forever, be advised that I still have copies of such letters dating almost ten years back (via my old *DTACK Grounded* newsletter). It was Pascal that was going to last forever back then. PJ

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

George Brandt,
programmer for KBHK-TV, San Francisco,
and BRIEF user since 1987.

Wayne Ratliff,
creator of dBASE,
president of Ratliff Software Productions,
and BRIEF user since 1986.

See us at Comdex
Hilton booth
#H8800

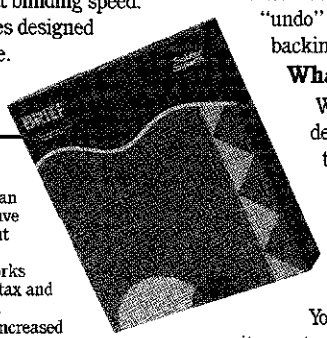
Of course they need completely different kinds of editors. That's why they both use BRIEF.

Let's face it. There's no such thing as two programmers who think alike. Or who work alike.

Because everybody's got their own style, their own habits. Their own quirky little requirements. As well they should. So, where are you going to find one editor that satisfies everyone? Simple. Look inside a BRIEF® box.

It's the right editor, right out of the box.

What you'll find inside is an enormously powerful, astonishingly easy, and highly-evolved program editor. One that works at blinding speed. And that's loaded with features designed to help you be more productive. One guaranteed to give you excellent results within one short hour of booting up the first time.



You'll find features like automatic indenting, infinite windowing, and powerful regular expression searching that goes well beyond the ordinary text-matching you're probably used to.

You'll also find the features that made BRIEF famous. Including our near-miraculous UNDO, that doesn't just un-delete your last keystroke. It actually reverses the effect of any command. So you can explore, experiment, even make gigantic mistakes with hideous consequences, and just "undo" them. And then continue to undo, backing-up through 300 steps.

What you want is what you get.

What you'll also find is an editor that delights in adapting itself to you. With the simple, menu-driven SETUP program, you can quickly customize indenting style, support for any of 36 compilers, keyboard configuration, rate of keyboard auto-repeat, and much more.

You can even edit BRIEF's macros to suit your taste. Or write your own in a snap.

Computer Language said that BRIEF is capable of "almost complete customization." And with new release 3.0, BRIEF is even more powerful and flexible.

For every programmer, it's a different BRIEF.

At last count, there were over 52,000 copies of BRIEF in the field, making life easier and working time more productive for all kinds of programmers—working in all kinds of languages.

Programmers as different as Wayne Ratliff is from George Brandt. Which means there are over 52,000 different ways of working, and BRIEF fits them all.

Why not get BRIEF going to work for you? There's no risk. In fact, if you're not completely satisfied that BRIEF is exactly the editor you've been looking for, just return it within 30 days, and we'll return your money. No questions asked.

New BRIEF 3.0 highlights

- With our new C-like macro language, CBRIEF, C programmers can write macros on the fly and not have to switch between C and a different macro language.
- The new source level debugger works with both the original BRIEF syntax and the new CBRIEF macro language.
- Maximum lines per file has been increased to 4 billion.
- We've added to and improved template editing and smart indenting for ADA, Cobol, BASIC, FORTRAN, Modula-2, Pascal, and C.
- We've added editable, multiple keystroke macros that can be saved and restored.
- You can even save and restore your window set-ups.

Plus the features that made BRIEF famous.

- 300-level UNDO safety net
- Flexible windowing, unlimited quantity
- Unlimited number of files
- File size limited only by disk space
- Regular expression search and replace
- BRIEF's LISP-like macro language
- Toll-free tech support

BRIEF 3.0

The Programmer's Editor

Solution Systems™
541 Main St., Suite 410
South Weymouth, MA 02190
FAX 617-337-7719

Try BRIEF at our risk. $249 plus $6 shipping (U.S. and Canada). 30-day money-back guarantee.

Name _____

Company _____

Address _____

City _____ State _____ Zip _____

Phone _____

Paying by ☐ Check ☐ MC ☐ VISA

Card # _____ Exp. date _____

Signature _____

Disk size ☐ 5-1/4" ☐ 3-1/2" Format ☐ DOS ☐ OS/2

Mail this coupon or call toll-free today.

800-821-2492

Extension 496

Massachusetts & outside U.S. call 617-337-6963

Circle **47** on Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Limited time offer to owners of Microsoft C

Borland's Turbo C++ Professional "Everything you always wanted in a C compiler and more"

—Reprinted from *PC Magazine*, August 1990

Copyright ©1990 Ziff Communications Company

Special upgrade offer for Microsoft® C owners!

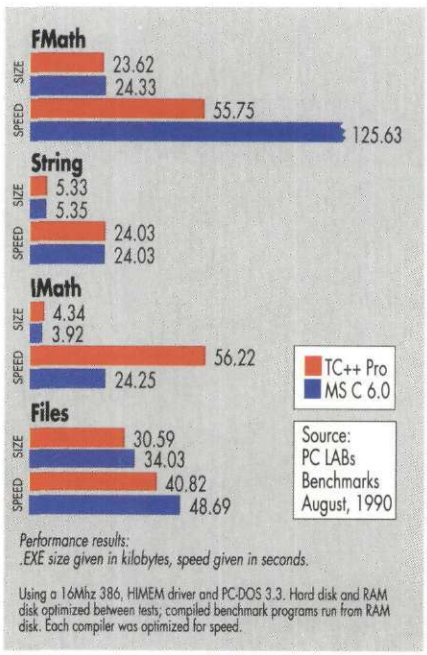
Are you ready for an upgrade that conforms 100% to ANSI C, gives you C++, doesn't skimp on documentation, features a fast, reliable programming environment, and provides the complete toolset necessary to maximize your productivity? Then it's time to make the move to Borland.

And the time to move is now. We're offering Turbo C++ Professional to owners of Microsoft C or any PC-based C or C++ compiler for only $149.95.*

Compare the features.

COMPILER	MSC 6.0	TC++ Pro
C++	NO	YES
Full ANSI C	NO	YES
Transparent overlay manager	NO	YES
Complete printed documentation	NO	YES
ENVIRONMENT		
Overlapping windows	NO	YES
Mouse support	YES	YES
Integrated debugging	NO	YES
TOOLS		
Debugger	YES	YES
Profiler	NO	YES
Macro assembler	NO	YES

Turbo C++ Professional beats Microsoft C 6.0.



Performance results:
.EXE size given in kilobytes, speed given in seconds.

Using a 16Mhz 386, HIMEM driver and PC-DOS 3.3. Hard disk and RAM disk optimized between tests; compiled benchmark programs run from RAM disk. Each compiler was optimized for speed.

"... if you want an excellent ANSI compiler with integrated environment, overlay linkage, profiler, assembler, and an award-winning debugger—not to mention C and C++ support—look no further: Turbo C++ Professional is here."

—*PC Magazine*, August 1990

Step up now to Turbo C++ Professional for only $149.95

That's half the suggested retail price of $299.95.

If you're not convinced that Turbo C++ Professional is the best upgrade to Microsoft C 5.1, just return the product for a full refund.

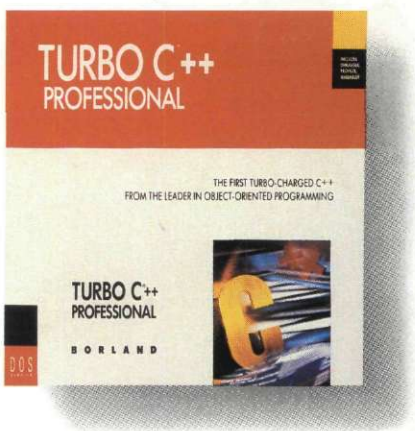
To order

SEE YOUR DEALER

(bring proof of ownership)

or call now

1 (800) 331-0877



BORLAND

Makers of Turbo C++, Turbo Pascal,® Paradox,® Quattro® Pro and Sidekick®

Code: MC12

*Proof of ownership (an original manual page or disk from any version) is required. Offer expires December 31, 1990. Offer good in U.S. and Canada only. Special discounts for registered Turbo® C owners are available from Borland. Mail orders to: Borland International, Inc., P.O. Box 660001, Scotts Valley, CA 95067-0001. Add $5.00 for shipping and handling. Residents in CA, CT, GA, IL, MA, MI, NY, OH, PA, TX, VA and WA please add appropriate sales tax. For orders outside the U.S., call (408) 438-5300. Turbo C++ and Turbo Language are trademarks of Borland International, Inc. Copyright ®1990 Borland International, Inc. All rights reserved. BI 1382

Circle 3 on Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED